Universidad Autónoma de Madrid
Escuela Politécnica Superior

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Trabajo Fin de Grado

DESARROLLO DE UN MARCO DE TRABAJO PARA LA RECONSTRUCCIÓN DE IMÁGENES A PARTIR DE CÁMARAS DE EVENTOS

Autor: Miguel Lorente Peinado
Tutor: Pablo Carballeira López
Ponente: José María Martínez Sánchez

Junio 2021

DESARROLLO DE UN MARCO DE TRABAJO PARA LA RECONSTRUCCIÓN DE IMÁGENES A PARTIR DE CÁMARAS DE EVENTOS

Autor: Miguel Lorente Peinado
Tutor: Pablo Carballeira López
Ponente: José María Martínez Sánchez

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2021

Trabajo parcialmente financiado por el Gobierno de España bajo el proyecto
TEC2017-88169-R (MobiNetVideo)

Resumen

Resumen

Las cámaras de eventos son sensores bioinspirados que monitorizan asíncronamente pixel a pixel los cambios de intensidad logarítmica y generan un flujo de eventos que codifica la marca de tiempo, la posición y la polaridad del cambio de intensidad, a diferencia de las cámaras convencionales que capturan una secuencia de fotogramas a una velocidad fija. Los sensores de visión basados en eventos han llamado la atención de la comunidad investigadora los últimos años debido a las ventajas que ofrecen frente a las cámaras convencionales para el desarrollo de aplicaciones de visión artificial, entre sus propiedades destacamos: alto rango dinámico, baja latencia, alta resolución temporal y bajo consumo de energía. Sin embargo, requieren nuevos métodos para procesar la señal de eventos, lo que supone un cambio de rumbo frente a los métodos tradicionales basados en fotogramas. Aún son pocas las cámaras disponibles en el mercado y su precio es considerablemente alto, además son pocos los *datasets* masivos que podemos obtener para la investigación y el desarrollo de nuevos algoritmos. No obstante, se han creado simuladores de eventos que nos permite generar flujos de eventos sintéticos a partir de secuencias de fotogramas. Por lo tanto, hemos creado un marco de trabajo para la reconstrucción de imagen donde evaluar la calidad de la reconstrucción en secuencias con ruido de eventos y en secuencias de eventos simuladas. Hemos contribuido creando una escala de nivel de ruido para evaluar la calidad de reconstrucción, usando los tipos de ruidos: añadir eventos, eliminar eventos, ruido temporal y ruido espacial, comprobando como el ruido más influyente en la calidad de reconstrucción es el ruido espacial y que nuestro reconstructor es más robusto frente a ruido temporal de eventos. También hemos propuesto un método de evaluación del reconstructor basado en secuencias de eventos simulados, donde hemos demostrado que ESIM no es un simulador adecuado para tareas de reconstrucción de imagen de intensidad basado en sítesis de vídeo, dadas las distorsiones de los eventos sintéticos.

Palabras Clave

Visión basada en Eventos, Cámara de Eventos, Simulador de Eventos, Reconstrucción de Imagen de Intensidad, Sítesis de Vídeo, Ruido de Eventos.

Abstract

Event-cameras are bio-inspired sensors that asynchronously report per-pixel log-intensity changes and outputs a stream of events encoding the timestamp, the position and the log-intensity change polarity, unlike conventional cameras that takes a frame sequence at a fixed rate. Event-based vision sensors have become very attractive over the last years because of their advantages over conventional cameras that makes them very useful for computer vision applications, among their properties we highlight high dynamic range, low latency, high temporal resolution and low power consumption. But they need novel methods to process their output, what suppose a paradigm shift for conventional frame-based methods. Event-cameras remain scarce and expensive, they also do not have massive datasets for research and novel applications development. However, there have been developed event simulators that allows us to generate synthetic event streams from frame-based sequences. Thus, this motivates us creating an image reconstruction pipeline, where we evaluate the reconstruction quality in noisy event streams and in simulated event streams. We have contributed creating a noise level scale to evaluate the reconstruction quality. Using four error types: add events, remove events, temporal noise and spatial noise, we have realized that spatial noise is the one that most influences in our image reconstructor, while the reconstructor is more robust against spatial noise. We have also proposed a reconstruction quality evaluation based in simulated event sequences, where we have demonstrated that ESIM is not the right simulator for intensity image reconstruction tasks based on video synthesis, because of its synthetic events distortions .

Key words

Event-Vision, Event-Camera, Event Simulator, Intensity Image Reconstruction, Video Synthesis, Event Noise.

Agradecimientos

En primer lugar quiero dar las gracias a mi familia y amigos que me han acompañado y apoyado en toda esta etapa. A mis compañeros Fernando, Santi, Camilo, Jorge... por los maratones de estudio en la biblioteca y las explicaciones a cinco minutos del examen. Y sobretodo a Alba, que me ha motivado y soportado durante este último empujón.

No puedo dejar sin agradecer todo lo que me han enseñado estos años mis profesores y personal docente de la escuela, en particular a Pablo, por ofrecerme este trabajo que me ha supuesto un reto a superar.

Gracias de corazón.

Índice general

Lista de figuras

Lista de tablas

1 Introducción

Las cámaras basadas en eventos son un tipo de sensor bioinspirado en el sistema visual humano (la primera cámara de eventos, *Silicon Retina* [9]). Aunque aún es una tecnología inmadura, presentan unas características que las hacen muy versátiles en el campo de la visión artificial y la neurorobótica. En este capítulo veremos una introducción al trabajo, centrándonos en el por qué de éste y el enfoque que le hemos dado.

1.1. Motivación

Las cámaras basadas en eventos son sensores bioinspirados que miden los cambios de intensidad de luz en escala logarítmica[1] pixel a pixel y de forma asíncrona. A diferencia de las cámaras convencionales que toman el valor de pixel síncronamente creando fotogramas a una tasa constante. La señal de pixel se denomina evento y se envía a la salida, lo que produce que la salida sea un flujo de eventos. Dado que la señal de salida de una cámara de eventos es diferente a la de una cámara convencional, los algoritmos de visión artificial de las cámaras convencionales no sirven. Por lo que, será necesario adaptarlos o desarrollar nuevos. Las cámaras de eventos ofrecen la posibilidad de desarrollar tareas que las cámaras convencionales no pueden o las desarrollan con mayor dificultad. Las ventajas que nos ofrecen las cámaras de eventos son: resolución temporal muy alta, baja latencia, rango dinámico muy alto y bajo consumo de energía.

El entorno de las cámaras de eventos es una tecnología aún inmadura, prueba de ello es que llevan comercializándose desde 2008 y son pocas las compañías que se dedican a ello. Los precios son elevados y son pocos los modelos disponibles, además aún son pocos los *datasets* masivos de secuencias grabadas con cámaras de eventos, como sucede en cámaras convencionales. Todo esto complica la investigación y el desarrollo de algoritmos de procesamiento de eventos. Para hacer frente a este problema se han creado diversos simuladores de cámaras de eventos para poder investigar en nuevos algoritmos y aplicaciones sin la necesidad de invertir en una cámara. Los simuladores se encargan de convertir una secuencia de vídeo obtenida de una cámara convencional en un flujo de eventos sintéticos.

Vistas las ventajas que ofrecen las cámaras de eventos y la posibilidad de investigar sin invertir una gran cantidad de dinero. La motivación pricipal de este trabajo será analizar las

[1] A partir de ahora para referirnos a la intensidad de luz logarítmica hablaremos de luminosidad.

posibilidades que nos ofrecen estos simuladores para el desarrollo de algoritmos de cámaras de eventos. Creando así un marco de trabajo para el procesamiento de señales de eventos.

1.2. Objetivo y enfoque

El objetivo principal de este trabajo es hacer una aproximación a la tecnología y funcionamiento de las cámaras de eventos. Dentro de las aplicaciones posibles, enfocaremos el trabajo en los algoritmos de reconstrucción y síntesis de video a partir de eventos y en la obtención de secuencias de eventos a partir de los simuladores. Para ello realizaremos las siguientes tareas:

1. Realizar un análisis detallado de la evolución de la tecnología de las cámaras de eventos. Estudio de la tecnología y funcionamiento de las cámaras de eventos, tipos de cámaras de eventos (DVS, ATIS, DAVIS) y las ventajas que nos ofrecen.
2. Implementar una cadena de procesamiento de señales de eventos, para evaluar los desafíos del procesado de eventos. Haciendo un análisis de los simuladores disponibles y su puesta en funcionamiento, para obtener nuestros propios flujos de eventos
3. Teniendo el marco de trabajo para el procesado de eventos ya desarrollado, analizar la influecia del ruido de eventos en la reconstrucción de vídeo a partir de eventos y evaluar la calidad de la simulación de eventos.

1.3. Estructura

Hemos comenzado haciendo una introducción detallada del punto en el que se encuentra la tecnología de las cámaras de eventos (Cap. 2) donde hemos descrito su funcionamiento (Sec. 2.2), las ventajas que ofrecen (Sec. 2.3), los inconvenientes que podemos encontrar en la tecnología de las cámaras de eventos (Sec. 2.4), los tipos de cámaras de eventos que podemos encontrar (Sec. 2.5), las aplicaciones que tienen las cámaras de eventos y algunos de los algoritmos que se han desarrollado centrándonos en los algoritmos de reconstrucción de vídeo (Sec. 2.6) y finalmente los simuladores disponibles (Sec. 2.7).

En el Capítulo 3 comenzaremos describiendo el marco de trabajo sobre el que realizaremos las pruebas y los elementos que lo componen. Lo dividimos en dos partes, en la Sección 3.2 lo enfocaremos como un diagrama entrada/salida, describiendo brevemente los *datasets* de entrada, lo que esperamos a la salida de la cadena y las técnicas de medida con las que sacar conclusiones. Mientras que en la Sección 3.3 nos centraremos en los procesos internos de la cadena de procesamiento: simulación y reconstrucción.

Las pruebas realizadas y los resultados obtenidos los desarrollaremos en el Capítulo 4. Tras comentar las secuencias elegidas para las pruebas (Sec. 4.2), explicaremos el método seguido para encontrar los parámetros del reconstructor adecuados para nuestras secuencias (Sec. 4.3), analizaremos cómo afecta el ruido a la reconstrucción de vídeo (Sec. 4.4) y evaluaremos la calidad de la simulación (Sec. 4.5)

Finalmente, en el Capítulo 5 comentaremos las conclusiones finales del trabajo y el trabajo futuro, donde mencionaremos las posibilidades por explorar en el entorno de este trabajo y de las cámaras de eventos .

2

Estado del arte

2.1. Introducción

Durante los últimos años las tecnologías de las cámaras basadas en eventos han experimentado un incremento en sus aplicaciones y en el desarrollo de nuevos algoritmos de procesado de eventos. Las cámaras de eventos ofrecen muchas ventajas frente a las cámaras convencionales en el ámbito de la robótica y de la visión artificial [1], como veremos en los siguientes apartados con más detalle.

Una cámara basada en eventos es un sensor bio-inspirado, que trata de imitar el funcionamiento del sistema visual humano. Su salida es un flujo de pulsos asíncronos, como si de impulsos nerviosos se tratase. Dado que los algoritmos utilizados para aplicaciones de visión artificial están diseñados para cámaras convencionales, cuya salida es distinta, los sensores bioinspirados son una tecnología aún inmadura y costosa.

Gran parte de la información e investigación que se ha realizado desde la primera "*Silicon Retina*" hasta ahora ha sido recopilada por parte de sus desarrolladores y se encuentra organizada en un repositorio github[1]. Este repositorio *github* es de gran importancia para la comunidad científica, ya que aglomera toda la información relacionada con la visión basada en eventos de manera accesible, organizada y de código abierto.

Comenzaremos estudiando el funcionamiento de una cámara basada en eventos, el sensor de visión dinámica (DVS, a partir de ahora) y en qué consiste un evento. Continuaremos con los tipos de cámaras de eventos que podemos encontrar y los algoritmos y aplicaciones que se usan actualmente para las cámaras de eventos.

2.2. Principio de operación

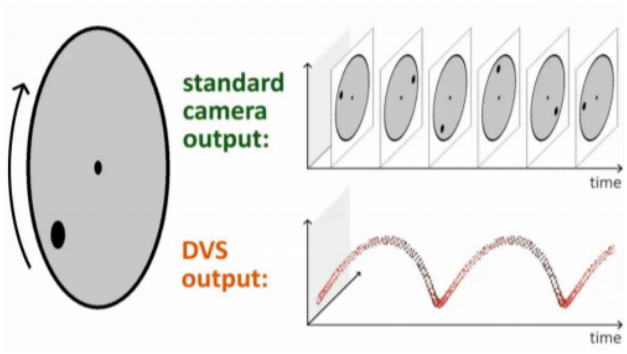
Una cámara basada en eventos se caracteriza por tener un sensor de visión dinámica (DVS), compuesto por una matriz de píxeles, que responde a los cambios de luminosidad en la escena de forma asíncrona e independiente para cada píxel. La respuesta de cada píxel se codifica creando un evento, el elemento principal del DVS [1]. A la salida de la cámara obtendremos un flujo

[1] https://github.com/uzh-rpg/event-based_vision_resources

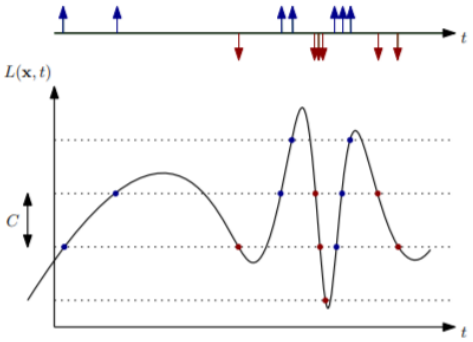
de eventos con una tasa de velocidad variable (Fig. 2.1a). A diferencia de una cámara basada en fotogramas[2] cuyos píxeles del sensor responden en síncronía a una tasa de fotogramas por segundo constante, obteniendo a la salida una secuencia de fotogramas. Esta diferencia podemos apreciarla en la Figura 2.1a.

Como hemos visto, un evento es la forma de codificar los cambios de luminosidad que detecta un píxel. Un píxel memoriza la intensidad logarítmica de luz cada vez que envía un evento. El píxel monitoriza la luminosidad y si pasa un umbral C superior o inferior al último valor registrado, se creará un nuevo evento. En la Figura 2.1b podemos ver un esquema de este proceso y observar como a mayor cambio de luminosidad, mayor número de eventos son generados. El píxel transmite el evento generado a un bus compartido con el resto de la matriz de píxeles. Por tanto, la señal de salida de la cámara es un flujo de eventos.

Un evento está compuesto por la localización *x,y* del píxel en la matriz del sensor, una marca de tiempo t y un bit de polaridad p. El bit de polaridad se puede considerar un valor 'ON/OFF' de la luminosidad, es decir, si el cambio de luminosidad es positivo *p=1*, en caso de ser negativo *p=-1*.



(a) Comparación entre la salida de un DVS y una cámara convencional. En esta figura podemos ver la respuesta de una cámara convencional (arriba) y un DVS (abajo) frente a un disco con un punto negro dibujado girando sobre su centro (izquierda). Figura adaptada de [8]



(b) Principio de generación de eventos. En esta gráfica observamos cómo cuando la intensidad (línea contínua negra) pasa el umbral C (líneas discontínuas horizontales) se genera un evento (puntos). El evento puede ser positivo (azul) o negativo (rojo), dependiendo si pasa el umbral superior o el inferior. Arriba aparace la salida representada en forma de pulsos asíncronos. Figura adaptada de [8]

Figura 2.1: Funcionamiento de un DVS.

En la Figura 2.1a vemos el resultado de grabar un disco blanco con un punto negro girando a una determinada velocidad. La salida de la cámara convencional capta una imagen estática a una velocidad constante, mientras que el DVS que tiene una resolución temporal mayor obtiene un flujo de eventos a una tasa variable.

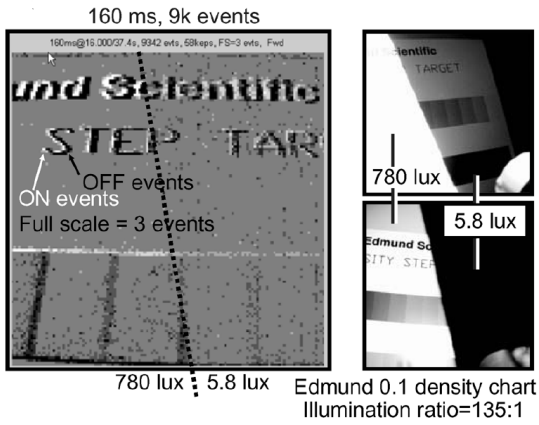
La salida depende de la cantidad de cambio de luminosidad, lo que se traduce como cantidad de movimiento en la escena. Por lo tanto, cuanto mayor sea el movimiento, más eventos se generan por segundo (Fig. 2.1b). Dado que la iluminación de una escena suele ser constante, los eventos se producen por superficies en movimiento sobre el campo de visión [1].

[2] A partir de ahora nos referiremos a las cámaras basadas en fotogramas como cámaras convencionales.

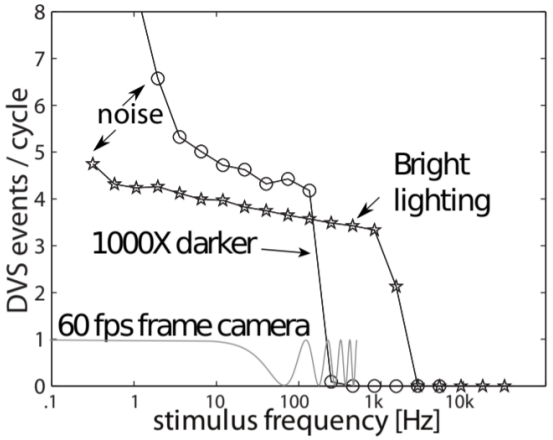
2.3. Características DVS

Las cámaras de eventos por el contrario tienen unas propiedades que prometen desbloquear una alta velocidad de percepción en estas situaciones. Estas propiedades son:

- **Alta Resolución Temporal:** La monitorización de luminosidad de los eventos con marca de tiempo son procesos rápidos que se llevan a cabo con una resolución de microsegundos [1]. El tiempo entre eventos es muy pequeño comparado con el tiempo entre fotogramas de una cámara convencional.
- **Baja Latencia:** Los píxeles trabajan asíncronamente, por lo que en cuanto se detecta una cambio de luminosidad se transmite, sin tener que esperar al resto de píxeles y sumar retardos. Por lo que un DVS reduce la latencia al máximo, del orden de microsegundos [1].
- **Alto Rango Dinámico:** El DVS puede pasar de operar en condiciones de luz muy bajas (0.1 lux) a muy altas (100klux) sin tener que reconfigurarlo [2], esto se debe a la comprensión logarítmica de la intensidad de luz. Además como los píxeles son asíncronos y cada uno ajusta su umbral, se puede obtener a la vez elementos de el entorno a oscuras e iluminados como se muestra en la Figura 2.2a. El rango dinámico de una cámara convencional puede llegar a 60dB, mientras que un DVS supera los 120dB [7] [1] (ver Fig. 2.3).
- **Gran Ancho de Banda:** Los píxeles DVS son muy rápidos pero aún así tienen un ancho de banda finito. En condiciones de buena iluminación un píxel puede tener un ancho de banda de 3kHz mientras que en condiciones de muy poca luz se reduce a 300Hz. Lo que quiere decir que incluso con muy poca luz tiene un ancho de banda mayor que una cámara convencional de 60fps (30Hz). Como podemos ver en la Figura 2.2b que es el resultado de un experimento DVS [2] y adaptado en [1].



(a) Comparación rango dinámico. Con una única respuesta de un DVS captamos a la vez la zona a oscuras y la iluminada (iquierda). Con una cámara convencional es son necesarias dos respuestas con distintas sensibilidades para poder capta ambas zonas. Figura adaptada de [2].



(b) Comparación ancho de banda. El ancho de banda de una cámara convencional (línea contínua) frente al ancho de banda de un DVS en condiciones de baja iluminación (línea con círculos) y alta iluminación (línea con estrellas). Figura adaptada de [1].

Figura 2.2: Propiedades DVS.

- **Bajo Consumo y Almacenamiento:** Dado que el DVS sólo transmite cuando se produce un cambio de luminosidad, automáticamente esta eliminando información redundante [1]. Por lo que gasta menos energía y se necesitan menos recursos para almacenar la señal de salida.

2.4. Inconvenientes

La gran mayoría de la investigación realizada en visión artificial hasta ahora ha estado centrada en las cámaras convencionales (basadas en fotogramas) [7]. Y en estos últimos años se ha conseguido aplicar comercialmente en drones, coches, aspiradoras... Pero como hemos visto en la Sección 2.3, las características de un DVS se asemejan las que requieren las nuevas aplicaciones de robótica. Por lo que pueden llegar a ser el futuro de algunas aplicaciones de visión artificial y de la neurorobótica.

Por ahora es una tecnología aún inmadura, ya que el funcionamiento del sensor (Sec. 2.2) es distinto al de una cámara convencional y hasta ahora tienen una resolución muy baja. Dado que los algoritmos convencionales de visión artificial están pensados para cámaras basadas en fotogramas, es necesario un procesado nuevo y complejo para desbloquear el potencial de los eventos [1].

El desarrollo de nuevos algoritmos y aplicaciones para el procesado de eventos avanza lentamente. Esto se debe a que las cámaras basadas en eventos son caras y poco comunes, es decir, son pocas las empresas que las comercializan y a un precio muy elevado. Por lo que, son pocos los equipos o laboratorios de investigación que tienen a su alcance un DVS [8]. Frente a este problema se han creado datasets y simuladores de los que hablaremos más adelante. En la Figura 2.3 se muestra una comparación de los sensores que ofrecen algunas de estas empresas.

	Supplier	iniVation			Prophesee				Samsung			CelePixel		Insightness
	Camera model	DVS128	DAVIS240	DAVIS346	ATIS	Gen3 CD	Gen3 ATIS	Gen 4 CD	DVS-Gen2	DVS-Gen3	DVS-Gen4	CeleX-IV	CeleX-V	Rino 3
Sensor specifications	Year, Reference	2008 [2]	2014 [4]	2017	2011 [3]	2017 [67]	2017 [67]	2020 [68]	2017 [5]	2018 [69]	2020 [39]	2017 [70]	2019 [71]	2018 [72]
	Resolution (pixels)	128 × 128	240 × 180	346 × 260	304 × 240	640 × 480	480 × 360	1280 × 720	640 × 480	640 × 480	1280 × 960	768 × 640	1280 × 800	320 × 262
	Latency (µs)	12µs @ 1klux	12µs @ 1klux	20	3	40 - 200	40 - 200	20 - 150	65 - 410	50	150	10	8	125µs @ 10lux
	Dynamic range (dB)	120	120	120	143	> 120	> 120	> 124	90	90	100	90	120	> 100
	Min. contrast sensitivity (%)	17	11	14.3 - 22.5	13	12	12	11	9	15	20	30	10	15
	Power consumption (mW)	23	5 - 14	10 - 170	50 - 175	36 - 95	25 - 87	32 - 84	27 - 50	40	130	-	400	20-70
	Chip size (mm^2)	6.3 × 6	5 × 5	8 × 6	9.9 × 8.2	9.6 × 7.2	9.6 × 7.2	6.22 × 3.5	8 × 5.8	8 × 5.8	8.4 × 7.6	15.5 × 15.8	14.3 × 11.6	5.3 × 5.3
	Pixel size ($µm^2$)	40 × 40	18.5 × 18.5	18.5 × 18.5	30 × 30	15 × 15	20 × 20	4.86 × 4.86	9 × 9	9 × 9	4.95 × 4.95	18 × 18	9.8 × 9.8	13 × 13
	Fill factor (%)	8.1	22	22	20	25	20	> 77	11	12	22	8.5	8	22
	Supply voltage (V)	3.3	1.8 & 3.3	1.8 & 3.3	1.8 & 3.3	1.8	1.8	1.1 & 2.5	1.2 & 2.8	1.2 & 2.8		1.8 & 3.3	1.2 & 2.5	1.8 & 3.3
	Stationary noise (ev/pix/s) at 25C	0.05	0.1	0.1	-	0.1	0.1	0.1	0.03	0.03		0.15	0.2	0.1
	CMOS technology (nm)	350	180	180	180	180	180	90	90	90	65/28	180	65	180
		2P4M	1P6M MIM	1P6M MIM	1P6M	1P6M CIS	1P6M CIS	BI CIS	1P5M BSI			1P6M CIS	CIS	1P6M CIS
	Grayscale output	no	yes	yes	yes	no	yes	no	no	no	no	yes	yes	yes
	Grayscale dynamic range (dB)	NA	55	56.7	130	NA	> 100	NA	NA	NA	NA	90	120	50
	Max. frame rate (fps)	NA	35	40	NA	NA	NA	NA	NA	NA	NA	50	100	30
Camera	Max. Bandwidth (Meps)	1	12	12	-	66	66	1066	300	600	1200	200	140	20
	Interface	USB 2	USB 2	USB 3		USB 3	USB 3	USB 3	USB 2	USB 3	USB 3			USB 2
	IMU output	no	1 kHz	1 kHz	no	1 kHz	1 kHz	no	no	1 kHz	no	no	no	1 kHz

Figura 2.3: Comparación de sensores comerciales o prototipos. Figura adaptada de [1].

2.5. Tipos de cámaras de eventos

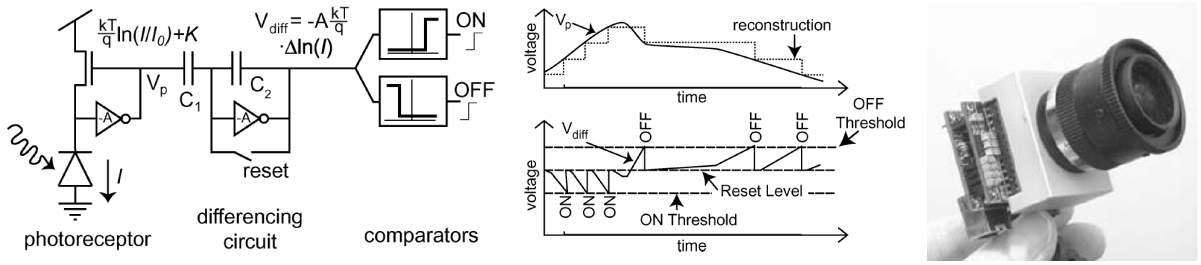
La primera cámara de eventos fue creada en 1992 por Mahowald y Mead [9]. Es un sensor bioinspirado en la retina humana, que produce un flujo de eventos a la salida usando el protocolo AER [10]. Son varias las cámaras de eventos que han sido desarrolladas desde entonces [2] [11] [12] [3]. Los modelos de cámaras de eventos que podemos observar dependiendo de su arquitectura son [13][3]:

2.5.1. *Dynamic Vision Sensor* (DVS)

El DVS (Fig. 2.4c) está basado en la primera *Silicon Retina*, donde cada píxel contiene un fotoreceptor [14] de tiempo contínuo conectado a un circuito de lectura. Este circuito, Fig. 2.4a, se resetea cada vez que el píxel es muestreado, es decir, cada vez que cambia de valor. En la Figura 2.4b observamos como se muestrea la luminosidad con el umbral, en la gráfica superior y en la

[3]Para más información de los nuevos sensores ver la Figura 2.3 en [1].

inferior como se resetea ese valor una vez se crea un evento [2]. Su principal inconveniente es su baja resolución, aunque los últimos DVS desarrollados por Samsung (DVS-Gen4) ha conseguido solucionar estos problemas consiguiendo una resolución de 1280x960.



(a) Arquitectura de píxel. (b) Principio de operación. (c) DVS 128 x 128.

Figura 2.4: *Dynamic Vision Sensor*. Figuras adaptadas de [2].

Son muchos lo problemas que se pueden resolver usando solamente los eventos DVS, pero a veces son necesarios valores estáticos de la escena. Por ello, se han desarrollado arquitecturas que combinan información estática (fotogramas) y dinámica (eventos), estas son ATIS [11] [15] y DAVIS [3] [16][4].

2.5.2. *Asynchronous Time Based Image Sensor* (ATIS)

Cada píxel del array de píxeles de la ATIS [11] se encuentra dividido en un subpíxel detector de cambio de luminosidad, que funciona como el DVS [2] comentado en la Sección 2.5.1. Y otro subpíxel encargado de medir la exposición, que es disparado cuando el DVS genera un evento y obtiene la intensidad absoluta. El disparador resetea un condensador a un voltaje alto, el cual se va descargando sobre un fotodiodo. Por lo que, cuanto más intensa sea la iluminación más rápido se descarga el condensador. Y solo los píxeles que cambien de intensidad proporcionarán su nuevo valor.

El sensor ATIS [11] [17] consigue un amplio rango dinámico (>120dB). El píxel final tiene el doble de tamaño que un píxel DVS, lo que difuculta la posibilidad de aumentar la resolución del sensor. En escenas con muy baja iluminación pueden tardar en generarse los eventos de intensidad. Los ATIS de última generación ya solucionan estos problemas y algunos son comercializados por Prophesse[5].

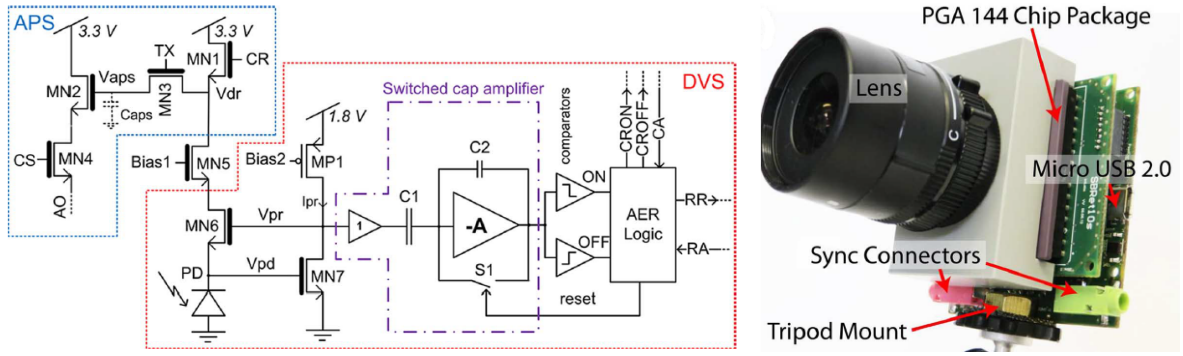
2.5.3. *Dynamic and Active Pixel Vision Sensor* (DAVIS)

Un píxel DAVIS [3] combina un píxel APS [18] convencional con un píxel DVS [2]. El fotodiodo del DVS se comparte entre píxeles, Fig. 2.5a, ocupando menos área de píxel. Por lo tanto, el píxel DAVIS tiene menor tamaño que el ATIS.

Los fotogramas de intensidad del sensor APS pueden obtenerse a tasa constante o según lo requieran los píxeles DVS. La lectura de los píxeles APS tienen un rango dinámico limitado (55dB) y su valor será redundante si no cambia la escena, mientras que los píxeles DVS mantienen su alto rango dinámico (>120dB) [3] [16].

[4]Dado que los sensores ATIS y DAVIS contienen DVS, también nos referimos a ellos cuando generalicemos con el término DVS.

[5]https://www.prophesee.ai/buy-event-based-products-2/



(a) Arquitectura de píxel.

(b) DAVIS 240 x 180.

Figura 2.5: *Dynamic and Active Pixel Vision Sensor*. Figuras adaptadas de [3]

2.6. Algoritmos y aplicaciones

Cómo explicamos en la Sección 2.3, las cámaras de eventos pueden llegar a revolucionar el campo de la neurorobótica debido a las ventajas que ofrecen frente a la cámaras convencionales, por lo que son muchos los algoritmos y aplicaciones que han sido desarrollados para tareas de visión artificial con cámaras de eventos. Incluso se han adaptado algoritmos para cámaras convencionales al entorno de las cámaras de eventos. Estos son algunos de los algoritmos y aplicaciones desarrollados, agrupados por tipos de tareas [1].

2.6.1. Detección y Seguimiento de Características

Es una de las tareas más básicas de visión artificial. Con un DVS podemos conseguir más robustez que una cámara convencional debido a su alta resolución temporal, ya que es capaz de captar señal los instantes entre fotogramas. Hay muchas formas de tratar los eventos, agrupárlos es útil para el *tracking* de objetos sencillos, como pequeñas partículas en [19] o para monitoreo y vigilancia de tráfico [20] [21]. Para objetos más complejos se realiza un procesado evento a evento, y se aplican algoritmos de vecino más próximo para agrupar eventos que forman parte de un objeto [22]. Para sensores que combinan eventos y fotogramas (v. g. DAVIS) la tarea resulta más sencilla, ya que a partir de los fotogramas detecta el objeto o característica y mediante los eventos se encarga del seguimiento [23] [24].

2.6.2. Estimación de Flujo Óptico

Esta tarea consiste en medir el movimiento píxel a píxel en el plano, reflejando los cambios en la imagen debido al movimiento de objetos en la escena. Con cámaras convencionales se obtiene el flujo óptico comparando dos fotogramas consecutivos [1], para un DVS es menos costoso. En [25] se hace posible mediante una secuencia de eventos y un fotograma de un DAVIS, ya que los eventos por sí solos no aportan suficiente información.Agrupando los eventos obtenemos una representación de los bordes de los objetos, por lo que una vez agrupados será más fácil calcular el movimiento en la escena y por consiguiente la velocidad de flujo óptico [26]. Para esta tarea hay multitud de algoritmos bio-inspirados, como en el caso de [27] qué también usa una red neuronal de impulsos, con la que detecta los patrones de movimiento. La estimación de flujo óptico es una tarea costosa computacionalmente, que ha crecido con el auge del *deep learning* dando lugar a algoritmos basados en eventos como [28] [29].

2.6.3. Reconstrucción 3D

La estimación de profundidad es la base de la reconstrucción 3D y es una tarea que dependerá del tipo de escenario y la configuración de cámaras que usemos. Las cámaras de eventos siguen el mismo enfoque que las cámaras convencionales para esta tarea, aunque con la ventaja de la señal de eventos. Para configuración estéreo se usan dos cámaras de eventos: **Reconstrucción Estéreo Simultánea** las cámaras estan sincronizadas. Trata encontrar la similitud de los eventos de ambos sensores y luego localizar cada punto 3D en escenas estáticas [30] [31]. **Panoramas Multiperspectiva** trata el mismo problema que reconstrucción estéreo simultánea, pero con dos cámaras no sincronizadas [32] [33]. **Estimación Estéreo de Profundidad para SLAM**[6] presenta una solución al problema de reconstrucción 3D estéreo en una escena en movimiento [34], útil para aplicaciones de robots y coches autónomos. Para configuración monocular se usa una única cámara: **Estimación de Profundidad Monocular** recrea una reconstrucción 3D de la escena integrando los eventos de una cámara en movimiento, requiere la información del movimiento de la cámara en el tiempo [35] [36]; **Estimación de Profundidad con Luz Estructurada** consiste en arrojar luz sobre la escena y medir la respuesta con un DVS [37], este método podemos catalogarlo como activo ya que interfiere en la escena, a diferencia del resto (pasivos).

2.6.4. Estimación de Posición y SLAM

SLAM está compuesto por tareas de más bajo nivel, y trata de estimar el movimiento 6-DOF[7] en un entorno natural en tres dimensiones. De entre las tareas de visión artificial SLAM es la que más cuesta adaptar al entorno de las cámaras de eventos, ya que la mayoría de los algortimos para cámaras convencionales nos son aplicables al entorno de los eventos. Por lo que requiere un cambio de enfoque, se han implementado en dos sentidos: El **Seguimiento de la Cámara** consiste en identificar su posición con la información visual que recibe del sensor en un entorno recreado [38] o un entorno natural [39] [40]; mientras que el **Seguimiento y Mapeo** realiza la misma tarea pero de objetos del entorno [36] [41].

2.6.5. *Visual Inertial Odometry (VIO)*

Es una tarea única para señales de eventos, las cámaras convencionales se usan para *Visual Odometry (VO)* que obtiene peores resultados que las señales de eventos. Consiste en combinar los datos asícronos de un DVS con los síncronos de una unidad de medición inercial (IMU) [1], para (dos tipos de sistemas): La mayoría de algoritmos VIO usan **VIO basado en características**, consiste en detectar y monitorizar características con los eventos mediante algoritmos de la Sección 2.6.1 y posteriormente combinar la trayectoria con la información del IMU[8] [42] [43]. **VIO basado en error de reproyección** desarrollado en [44]. Fusiona los eventos y la información del IMU en tiempo contínuo, lo que permite un procesado más rápido que el método basado en características.

[6] SLAM (*Simultaneous Localization and Mapping* o Localización y Mapeo Simultáneos): Otra de las tareas que engloba la visión atificial.

[7] 6-DOF (*Six Degrees Of Freedom* o Seis Grados de Libertad): hace referencia a la capacidad de moverse en un espacio tridimensional y rotar perpendicularmente sobre cada uno de los ejes

[8] IMU (*Inertial Measurement Unit* o Unidad de Medición Inercial): es un dispositivo que mediante acelerómetros y giroscópios mide la orientación, velocidad y aceleración.

2.6.6. Segmentación de Movimiento

La segmentación de objetos en movimiento es sencilla cuando la cámara es estática, como en [22] o [45]. Esta tarea se complica cuando la cámara también se encuentra en movimiento, donde es necesario diferenciar entre el movimiento de la cámara y los objetos de la escena. Glover y Bartolozzi [46] usan algoritmos de flujo óptico (Sección 2.6.2) para segmentación de objetos ya identificados previamente. Los eventos se agrupan en *clusters* que contienen los objetos en movimiento de la escena e introducen una compensación de movimiento para contrarrestar el movimiento de la cámara [47] [48].

2.6.7. Reconocimiento

Los algoritmos de reconocimiento han crecido en complejidad. Los objetos simples se pueden detectar comparando con plantillas de objetos predefinidos [49]. Para objetos más complejos se detectan sus características más sencillas y según su distribución un clasificador decide el tipo de objeto [50]. Con el desarrollo del *deep learning* el reconocimiento en secuencias de eventos ha avanzado, pero aún no ha llegado a alcanzar el nivel de desarrollo de los algoritmos para cámaras convencionales [1].

2.6.8. Reconstrucción de Imagen

Teniendo en cuenta que los eventos sólo contienen los cambios de intensidad son información visual no redundante, por lo que consideramos que son una forma de codificar la información visual. La tarea de reconstrucción de imagen a partir de la integración de eventos puede considerarse como decodificar el flujo de eventos [1]. Dada la alta resolución temporal de un DVS (Sec. 2.3), se pueden reconstruir imágenes a una tasa de fotogramas por segundo muy elevada ($>1200 fps$) [51]. La integración de los eventos sólo genera imágenes de incremento de luminosidad, como en [52] o [53], por lo que pueden ser necesarias imágenes de intensidad para reconstruir las partes estáticas del entorno. Esto implica que es posible convertir eventos a imágenes de intensidad, obteniendo así una secuencia de vídeo, y aplicarle algoritmos de visión artificial ya maduros [1]. Al igual que muchas de las actuales aplicaciones de visión artificial con cámaras convencionales, que comienzan haciendo un procesado de reconstrucción de imagen.

Son muchos los algoritmos de reconstrucción de información visual disponibles, y se agrupan dependiendo de la manera que afrontan esta tarea. En primer lugar, los algoritmos de **Reconstrucción de imagen de intensidad** usan únicamente eventos para la reconstrucción, como los citados anteriormente [51], [52] y [53]. El principal problema de la reconstrucción de imagen sólo con eventos es reconstruir la imagen completa, dado que un DVS no mide las intensidades absolutas. Por lo que, los algoritmos de **Síntesis de vídeo** proponen una reconstrucción a partir de un flujo de eventos y una secuencia de imágenes de intensidad [54] [55]. Ha quedado demostrado en algunos trabajos, como [56] o [57], que se obtiene una mayor calidad de reconstrucción mediante síntesis de vídeo, por lo que en este trabajo nos centraremos en esta manera de afrontar la reconstrucción de imagen.

Algoritmos de Síntesis de Vídeo

Entre los tipos de cámaras de eventos (Sec. 2.5) que obtienen flujos de eventos y fotogramas, ATIS es útil para reconstruir imágenes en escenas estáticas [58], pero dado que tiene defectos frente al movimiento rápido y su latencia incrementa en condiciones de baja iluminación, no es el sensor más adecuado para síntesis de vídeo. DAVIS en cambio mantiene una baja latencia en

los eventos y se adapta mejor a movimientos rápidos [54], por lo que es el sensor más elegido para esta tarea [56] [59] [55] [57].

En [60] se propone un modelo usando un DVS y una cámara convencional grabando la misma escena. Con la cámara convencional se obtienen las imágenes de intensidad de la escena estática, necesarias para completar el fondo de la escena. Mientras que el primer plano se reconstruye mediante los eventos y las imágenes de intensidad.

El trabajo expuesto en [54] define la arquitectura y funcionamiento básico de un reconstructor basado en síntesis de vídeo en tiempo real. Scheerlinck et al. [61] propone un método de predicción que puede funcionar con o sin imágenes de intensidad. En [56] se aplican algoritmos de estimación de profundidad y estimación de posición para lograr un mejor resultado. Para lograr una mejor calidad de reconstrucción [55] adapta algoritmos de *deblurring* usados para cámaras convencionales y [59] le sigue introduciendo algoritmos de predicción de fotograma e interpolación de fotogramas. Algunos trabajos como [59] y [57] también han aplicado *deep learning* a sus algoritmos, añadiéndole robustez aunque a su vez conlleva entrenar los algoritmos, lo que supone un problema teniendo en cuenta los pocos dataset de secuencias de eventos disponibles. Ahora veremos estos algoritmos más detalladamente:

1. ***Real-Time High-Speed Video Decompression*** [54] Es el primer algoritmo que trata la reconstrucción basada en síntesis de vídeo y establece el funcionamiento básico de este tipo de algoritmos en tiempo real, que se puede dividir en dos pasos:

   - En primer lugar, realiza una decodificación de los eventos, que según van llegando se añaden a una imagen de intensidad logarítmica o pseudo-intensidad.
   - Finalmente, cuando llega una nueva imagen de intensidad se estiman los pasos de intensidad (el umbral) de los eventos entre los fotogramas.

   Los resultados que obtiene el reconstructor son aceptables incluso en movimientos rápidos, aunque los objetos en movimiento dejan trazos de su trayectoria y se producen *lags*[9] entre imágenes.

2. ***Photorealistic image reconstruction*** [56] Emplea algoritmos de visión artificial (mapas de profundidad y estimación de posición) para conseguir una reconstrucción más fiel a la realidad. El algoritmo consta de cuatro pasos:

   - Se estiman los mapas de profundidad a partir de las imágenes de intensidad.
   - Se mapean los eventos entre los fotogramas de intensidad a las futuras nuevas imágenes o imagen de pseudo-intensidad, según [53].
   - Mediante odometría visual se estima la posición (6-DOF) de la cámara para cada una de las imágenes de pseudo-intensidad.
   - Se deforma la imágen según la posición de cada imágen de pseudo-intensidad y se fusiona con la imagen de pseudo-intensidad correspondiente.

   En cuanto a la reconstrucción en condiciones de movimientos bruscos, las imágenes de intensidad se desenfocan y se produce mucho ruido por lo que es dificil su reconstrucción. Además al usar los algoritmos de mapas de profundidad y posición aumenta el coste computacional y su latencia.

[9] Los *lags* son distorsiones producidas por retardos de la señal. En este caso se poducen de forma que los fotogramas reconstruidos conservan información visual de los fotogramas anteriores.

3. ***High Frame Rate Video Reconstruction*** [55] Los reconstructores basados en síntesis vistos hasta ahora consiguen mejores resultados que los reconstructores basados únicamente en eventos. Este es el primero que consigue afrontar el problema del desenfoque de movimiento (muy común en cámaras convencionales), implementando un algoritmo de *deblurring* llamado EDI (*Event-based Double Integral*) que se basa en otros algoritmos de cámaras convencionales. EDI recupera las imágenes latentes que se encuentran en una imagen con desenfoque producido por el movimiento y los eventos. Este algoritmo logra una mejor calidad que en los algoritmos ya descritos, aunque su rendimiento decae cuando cambian las condiciones de luz bruscamente y con la acumulación de ruido de eventos. En el marco de trabajo que definiremos más adelante utilizaremos este algoritmo de reconstrucción.

4. ***Video Synthesis*** [57] Sigue el modelo básico de decodificador usando imágenes de intensidad y eventos, implementado con un método de *deep learning* entrenado para combinar imágenes y eventos, y así estimar nuevas imágenes de intensidad entre fotogramas.

5. ***Event-driven Video Frame Synthesis*** [59] Es el reconstructor basado en síntesis más completo hasta ahora. Se inspira en ciertos aspectos de otros trabajos ya mencionados, como de [55] el *deblurring*. El algoritmo principal es el DMR (*Differentiable Model-based Reconstruction*) e implementa interpolación, predicción y *deblurring*. Introduce una nueva manera de agrupar eventos en la que cuando llega un nuevo evento, si su posición en las pseudo-imagen de intensidad ya tiene un píxel se mapea a una nueva pseudo-imagen, sino se mapaea a la actual. A diferencia de los anteriores reconstructores que agrupaban los eventos en un intervalo de tiempo definido para crear la pseudo-imagen de intensidad. El rendimiento de este algoritmo es mejor que los ya expuestos, pero para ello es necesario ajustar muchos parámetros, por lo que es necesario y aconsejable acompañarlo de una algoritmo de *deep learning* que habrá que entrenar.

2.7. Simuladores de cámaras de eventos

Cómo hemos visto en la Sección 2.4, los sensores DVS disponibles en el mercado son pocos y su precio es elevado. Por lo que, para investigar y desarrollar nuevos algoritmos y aplicaciones sin invertir dinero en un DVS, podemos usar *datasets* de secuencias de eventos, pero son muy pocos los que hay disponibles. Frente a este problema se han desarrollado distintos simuladores de eventos a partir de fotogramas. Un simulador DVS es una solución barata, pero no es una solución trivial [8]. Su funcionamiento básico generalmente consiste en obtener un flujo de eventos a partir de los fotogramas procedentes de una secuencia de vídeo de una cámara convencional (Fig. 2.6).

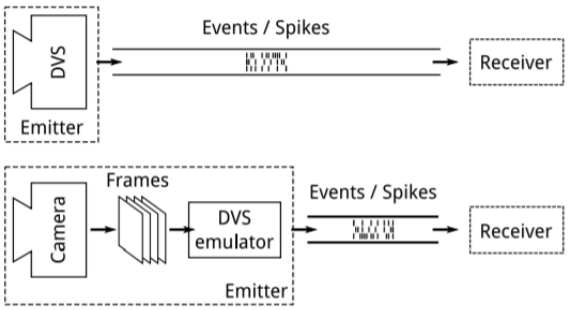


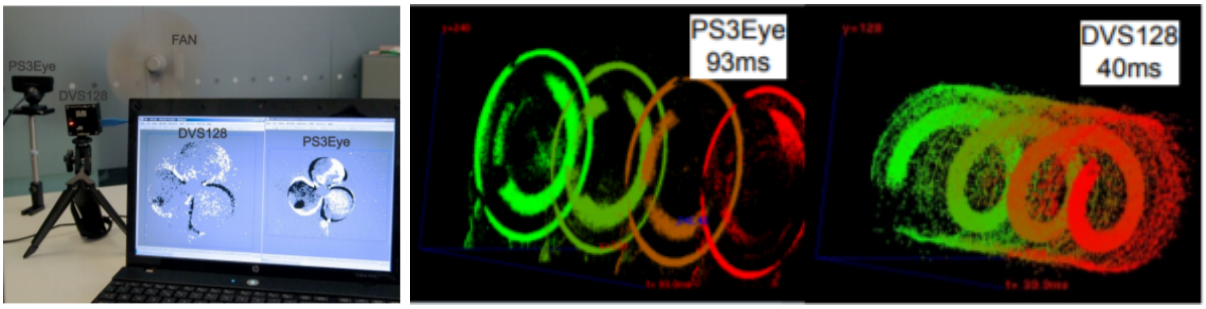
Figura 2.6: Diagrama DVS y emulador/simulador DVS. Figura adaptada de [4]

Aunque los simuladores tienen un mismo objetivo, que es simular eventos con la mejor calidad posible, encontramos distintos tipos de arquitecturas. La arquitectura y función básica de un simulador fue definido por primera vez [5], con el objetivo obtener un flujo de eventos en tiempo real y sin necesidad de un DVS. Otro dimulador que funciona en tiempo real es PIX2NVS [6]. Se han usado simuladores para aplicaciones neurorobóticas [62], donde se uas un simulador de eventos para dirigir un coche autónomo con una red neuronal de impulsos. Este simulador no implementa la asincronía de los eventos, por lo que no lo tendremos en cuenta en nuestro análisis. De entre los simuladores disponibles hay dos arquitecturas los que usan la aruitectura básica [5] [4] [6], que necesitan videos de alta tasa de fotogramas por segundo, y los que usan motores de renderizado 3D y posteriormente aplican el modelo básico [7] [63] [8], para obtener más imágenes de entre las originales para obtener un video de alta tasa de fotogramas por segundo y además obtener mapas de profundidad, IMU e imágenes de intensidad del DVS.

2.7.1. Emulador de Comportamiendo DVS (VSBE).

El objetivo del VSBE es mostrar cómo podemos emular la actividad de un DVS con cualquier cámara convencional barata. Su configuración, Figura 2.7a, consiste en una cámara convecional (PS3Eye), de alta tasa de cuadros por segundo, a la entrada. Cuyos fotogramas son procesados mediante el software para generar eventos con marca de tiempo sintética, como si fueran generados por un DVS [5]. El algoritmo para modelar el comportamiento de un DVS consiste en:

1. Mapear el valor de intensidad a su valor logarítmico. El mapeo es lineal para los valores pequeños próximos a cero y logarítmico para el resto de valores.
2. Compara si el valor logarítmico de píxel difiere con el del fotograma anterior, que ha sido almacenado en un array bidimesional. Si son iguales vuelve al mapeo.
3. Poner bit de polaridad ON/OFF a la salida. Dependiendo del cambio de intensidad logarítmica y el umbral.
4. Guarda el nuevo valor de píxel en el array bidimensional.



(a) Set-up (b) Prueba punto negro sobre disco blanco giratorio

Figura 2.7: Configuración del VSBE. Figuras adaptadas de [5]

La marca de tiempo de los eventos se genera interpolando linealmente entre el tiempo del último fotograma y el actual. Y depende del número de eventos creados entre dichos fotogramas. El umbral varía para cada píxel siguendo una distribución normal con la media y la desviacón típica del emulador.

Los resultados del simulador son muy similares al DVS en situaciones con una cantidad de movimiento medio como un ventilador girando despacio, Fig. 2.7a. Pero frente a un estímulo

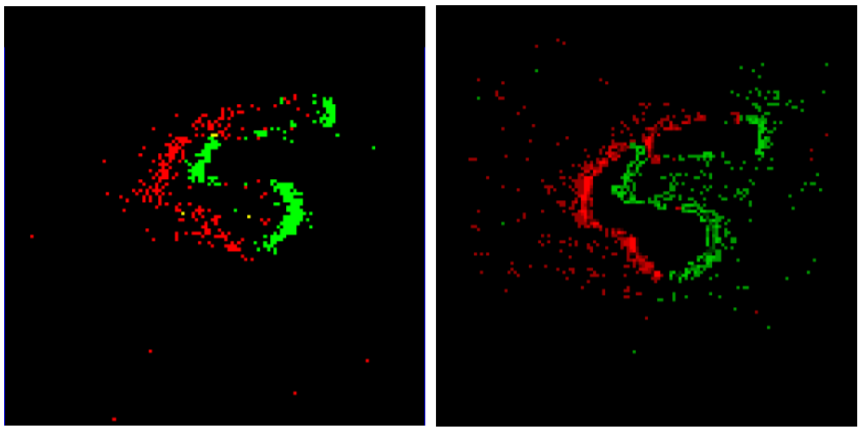
rápido sus limitaciones se hacen visibles. La Figura 2.7b es el resultado de grabar simultáneamente con un DVS y nuestra cámara convencional conectada al VSBE, un disco blanco girando sobre su eje con un punto negro dibujado. Para el DVS el punto genera una hélice de eventos en el eje temporal (Fig. 2.7b derecha). VSBE en cambio, capta una distorsión del punto en el tiempo entre fotogramas. Esto se debe a que el emulador solo puede generar una polaridad de píxel por cada fotograma (Fig. 2.7b izquierda).

2.7.2. pyDVS: Emulador DVS en tiempo real.

Su objetivo principal es ofrecer un sistema conversor de imágenes a eventos en tiempo real. Su funcionamiento básico compara en tiempo real el último fotograma (IMG) recibido con otro de referencia (REF), obteniendo así el de de diferencia (DIFF). A DIFF que contiene los posibles eventos se le aplica un umbral (threshold) constante, obteniendo así los eventos entre fotogramas, a los cuales se les marca con el tiempo interpolado entre fotogramas, como en VSBE [5], sección 2.7.1.

Las extensiones sobre el esquema básico que aporta pyDVS [4] son: **Codificación de tiempo** de los pulsos de salida, para no saturar el bus de salida. **Decaimiento de historial**, lo que añade robustez a la transmisión y permite al receptor recuperarse de fallos de transmisión a largo plazo. **Umbral adaptativo**, si un píxel no emite eventos en un tiempo el umbral se reduce, y al revés si emite muchos eventos en poco tiempo. E **inhibición local**, cuando píxeles cercanos tienen valores similares asumimos que tienen información redundante, por lo que emitiremos sólo el de mayor valor.

Los resultados de la simulación con pyDVS se pueden ver en la Figura 2.8, y salvo por algo de ruido debido a la configuración de ciertos parámetros, logra una simulación fiel al DVS en condiciones normales.



(a) DVS original (b) Resultado emulador DVS

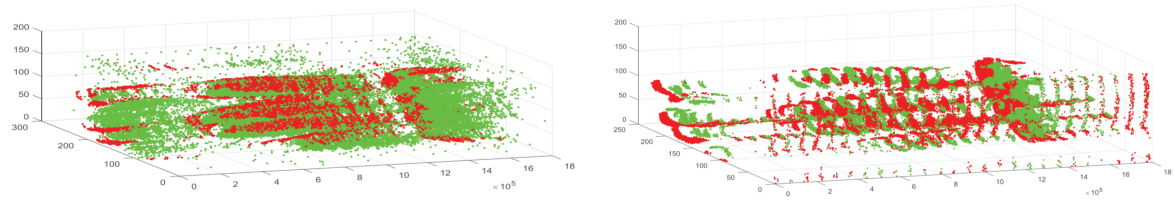
Figura 2.8: Comparación entre la secuencia DVS original y el resultado del emulador pyDVS. Figura adaptada de [4]

2.7.3. PIX2NVS

El software PIX2NVS sirve para generar flujos de eventos a partir de cualquier formato de vídeo convencional en el dominio de píxel [6]. Su objetivo es generar eventos los más parecidos a sensores como DAVIS (Sec. 2.5.3) o ATIS (Sec. 2.5.2).

El funcionamiento del simulador es común con otros simuladores, VSBE (Sec. 2.7.1). Aunque nos proporsiona tres métodos de asignación de la marca de tiempo de un evento, en el tiempo del

fotograma actual (eventos no asíncronos), interpolando linealmente o un tiempo aleatorio entre el fotograma actual y el anterior. Además al transformar el valor de píxel a evento, proporciona dos métodos de intensidad logarítmica o realce de contraste. Los resultados del simulador podemos verlos en la Figura 2.9 y más detalladamente en [6].



(a) Secuencia original

(b) Secuencia simulada

Figura 2.9: En la secuencia simulada se observa cómo los eventos se agrupan alrededor de los fotogramas (Verde/Rojo ON/OFF). Figura adaptada de [6].

2.7.4. Simulador DAVIS

Es un simulador de funcionamiento básico orientado a simular el comportamiento de una DAVIS, en concreto el IMU (acelerómetro y giroscopio) [7]. Dada una escena 3D virtual y la trayectoria de una DAVIS en movimiento, genera su correspondiente flujo de eventos, las imágenes de intensidad y los mapas de profundidad.

Mediante un software de renderizado se generan cientos de imágenes de intensidad durante la trayectoria. En la Figura 2.10 vemos como para cada píxel del sensor se realiza un seguimiento del último evento generado y comprobando si los cambios de intensidad superan el umbral. Combinando con la interpolación de tiempo entre las imágenes renderizadas nos permite crear marcas de tiempo continuas, como si fuesen eventos generados asíncronamente.

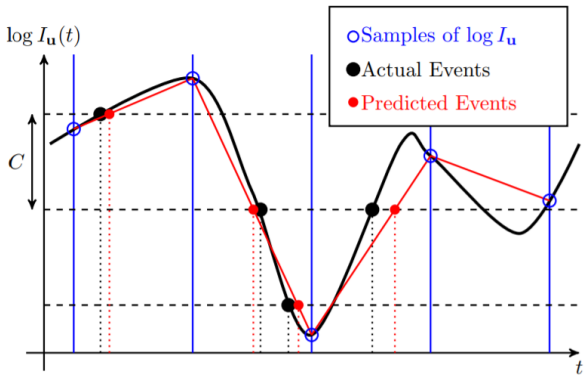


Figura 2.10: Seguimiento de intensidad e interpolación de tiempo.
Las líneas azules verticales muestran los instantes de cada fotograma, la línea contínua y los puntos negros muestran los eventos originales y la roja los simulados. La interpolación se produce linealmente entre fotogramas y al cruzar el umbral (C, línea negra discontínua). Figura adaptada de [7].

2.7.5. ESIM

ESIM hasta ahora es el simulador más usado en la literatura, por ello y por su rendimiento es el simulador que usaremos en nuestro marco de trabajo. Combina la estructura básica y un motor de renderizado 3D. La arquitectura de ESIM [8] podemos verla en la Figura 2.11. Une la arquitectura de un simulador básico con un software de renderizado, al cual le pasa la trayectoria de la cámara para obtener los mapas de profundidad y de movimiento de campo.

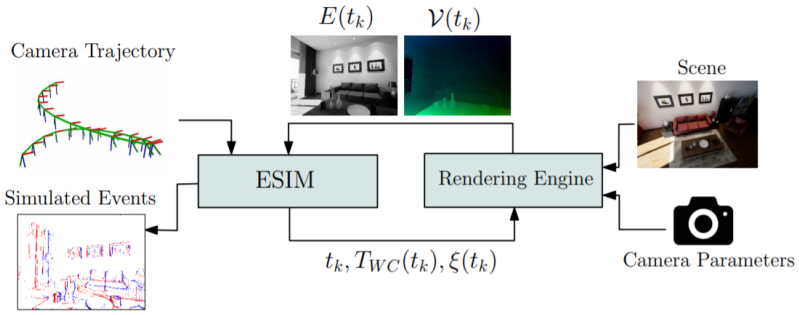
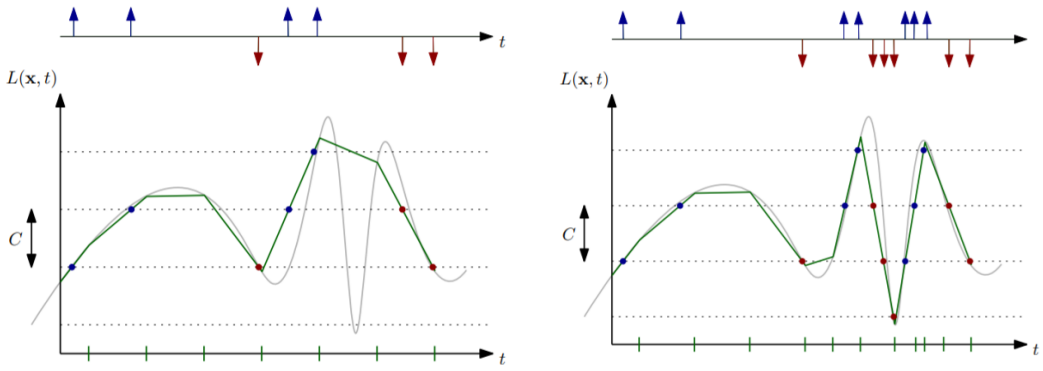


Figura 2.11: Arquitectura ESIM. Figura adaptada de [8].

El simulador usa el **sensor de trayectoria** para mapear a cada tiempo la posición, velocidad y aceleración. Y el **motor de renderizado** asigna a cada tiempo una imagen renderizada de la escena en la posición del sensor. Su particularidad se produce al acoplar el simulador con el renderizador, para un **renderizado adapativo**. El **renderizado adaptativo** consiste en muestrear los fotogramas adaptativamente, ajustando la frecuencia de muestreo en función de la predicción visual o la velocidad a la que cambia la escena (Fig. 2.12b). El **muestreo adaptativo** puede estar basado en cambio de brillo o en el desplazamiento de píxel. En la Figura 2.12 podemos observar la diferencia entre usar muestreo uniforme o adaptativo.



(a) Muestreo uniforme

(b) Muestreo adaptativo

Figura 2.12: Comparación entre muestreo uniforme y adaptativo. Figura adaptada de [8]. En la Figura 2.12b observamos como al variar la frecuencia de muestreo se consigue una simulación más fiel a la salida de una DAVIS.

2.7.6. *ExaRenderer*

Li et al. [63] proponen una cadena de procesos con el objetivo de renderizar imágenes foto-realistas y para crear un *dataset* que sirva para las nuevas aplicaciones del *deep learning* en visión artificial. Este trabajo está orientado a escenas en interiores, y se adapta distintos tipos de iluminación, tipos de objetivos, profundidades... para aplicaciones de SLAM.

El renderizador simula los efectos de cámara, como el desenfoque por movimiento. Representa la posición y orientación del sensor, también simula el acelerómetro y giroscopio del IMU, elementos que pueden perteneceer a un DVS. Además el simulador es capaz de generar un flujo de eventos a la salida, como el de una cámara de eventos. Renderizando a una alta tasa de fotogramas por segundo puede evaluar los cambios de luminosidad píxel a píxel y generar los eventos interpolando timestamps [63].

3

Marco de trabajo.

3.1. Introducción

El objetivo de este trabajo es hacer una aproximación al procesado de eventos, para ello implementamos una cadena de procesamiento de señal de eventos, que definiremos en este capítulo. Una vez implementada la cadena de procesamiento veremos como afecta el ruido de eventos en la reconstrucción de vídeo y evaluaremos la calidad del simulador.

En este capítulo vamos a ver el entorno y aplicaciones sobre las que se desarrolla este trabajo. En primer lugar lo veremos de forma general con un diagrama de entrada/salida, así como las técnicas de medida y comparación que usaremos para evaluar el simulador (Sec. 3.2). Posteriormente pasaremos a centrarnos más en detalle en los bloques que componen el entorno de trabajo: el simulador y el reconstructor de video seleccionados y la conexión entre estos (Sec. 3.3).

3.2. Diseño

En este apartado definimos un diagrama de bloque de entrada/salida general (Fig. 3.1), donde nos centramos en una explicación detallada del *dataset* original y el de prueba y como obtenemos las secuencias ruidosas y las simuladas a partir de las originales. También veremos el formato de salida del diagrama (Sec. 3.2.2) y de las técnicas de medida que usamos para comparar las distintas secuencias (Sec. 3.2.3).

3.2.1. Entrada

La entrada del diagrama general son los videos seleccionados del *dataset*, con secuencias originales de una DAVIS. Aunque en esta sección veremos también las secuencias ruidosas y las simuladas ya que forman parte de nuestro dataset y también son entrada para los bloques internos del diagrama.

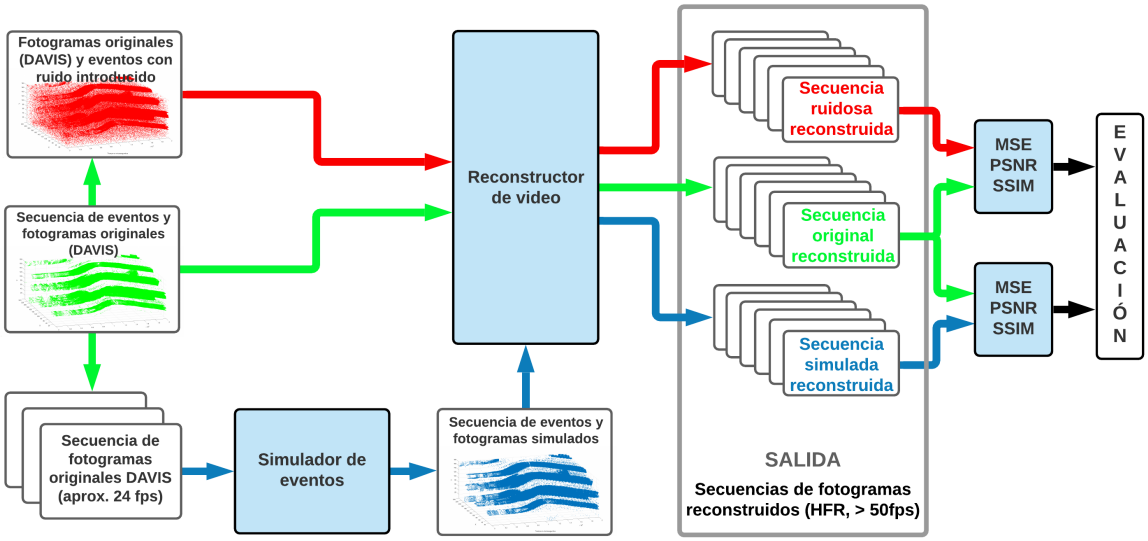


Figura 3.1: Diagrama de bloque general.

Dataset Original

El *dataset* original son un conjunto de secuencias obtenidas en su totalidad a partir de una DAVIS (Sec. 2.5.3) en diversas situaciones y condiciones de luz. Se encuentra en [7][1] y esta pensado para probar y evaluar tareas de estimación de posición, odometría visual y SLAM en el entorno de las cámaras de eventos.

Dataset de prueba

El *dataset* de prueba proviene del *dataset* original y es el que usaremos para nuestras pruebas y experimentos. Esta compuesto por tres tipos de secuencias: originales, ruidosas y simuladas.

Secuencias originales Las secuencias originales son las once seleccionadas del *dataset* compuestas por una secuencia de fotogramas a una tasa constante de alrededor de 24*fps* y un flujo de eventos DVS. El flujo de procesamiento por el que pasan se muestra en la Figura 3.1 con las flechas en verde.

Secuencias ruidosas Las secuencias ruidosas han sido obtenidas de un *dataset* del VPULab[2], que han sido obtenidas a su vez el *dataset* de secuencias originales [7]. En la Figura 3.1 podemos ver su recorrido de procesado mediante las flechas rojas. A cada una de las 11 secuencias seleccionadas, se les introdujeron 4 tipos de errores sobre los flujos de eventos, en 10 porcentajes distintos por cada tipo de error. Por lo que tenemos 440 secuencias ruidosas, en las que la secuecia de fotogramas siguen siendo los originales DAVIS. Los tipos de errores introducidos son:

- **Añadir Eventos:** Añade a la secuencia un porcentaje del total de eventos de la secuencia. Los eventos son generados aleatoriamente siguiendo una distribución uniforme, donde los porcentajes varían entre 0.5 % y 50 %.

[1] http://rpg.ifi.uzh.ch/davis_data.html
[2] http://www-vpu.ii.uam.es/webvpu/es/inicio/presentacion/

- **Eliminar Eventos:** Elimina de la secuencia un porcentaje del total de los eventos de la secuencia. Los eventos son generados aleatoriamente siguiendo una distribución uniforme, donde los porcentajes varían entre 0.5 % y 50 %.
- **Añadir Ruido en Tiempo *(ts)*:** Se aplica un desplazamiento temporal sobre la marca de tiempo de los eventos, que depende de un porcentaje del tiempo medio entre eventos. El desplazamiento sigue una distribución gaussiana, donde los porcentajes varían entre 5 % y 70 %.
- **Añadir Ruido en Espacio *(x,y)*:** Desplaza en el espacio x,y los eventos totales de la secuencia, es decir, cambia su posición en el sensor. El desplazamiento sigue una distribución gaussiana, cuya desviación es 3σ, donde σ es un porcentaje entre 0.25 % y 5 % de la diagonal del sensor.

Secuencias simuladas Las secuencias simuladas se obtienen a partir de los fotogramas de las secuencias originales, que forman una secuencia de video con una tasa de unos 24*fps* aproximadamente. La secuencia de video será la entrada del simulador, cuyo proceso definiremos en la Sección 3.3.1. A la salida del simulador obtendremos las secuencias simuladas, que estarán compuestas por un flujo de fotogramas y eventos sintéticos. El procesado de las secuencias simuladas se muestra mediante las flechas azules en la Figura 3.1.

3.2.2. Salida

A la salida del diagrama de bloque tendremos las secuencias de vídeo obtenidas a partir del reconstructor, para cada uno de los tipos de secuencia (originales, ruidosas y simuladas). Se muestran en el diagrama de bloque dentro del recuadro gris (Fig. 3.1) y posteriormente le aplicaremos la siguentes técnicas de medida (Sec. 3.2.3). Estas secuencias de vídeo tienen un alto número de fotogramas por segundo (HFR[3]), mayor a 50*fps*.

3.2.3. Técnicas de medida

Las siguientes técnicas de medida se usan para calcular errores o similitudes entre un resultado obtenido y una estimación previa, es decir, lo que esperamos. En nuestro caso usaremos estas fórmulas en su forma para comparación de imágenes, donde compara cada fotograma de video reconstruido píxel a píxel.

En primer lugar se aplicarán sobre las secuencias ruidosas, como estimador, y las secuencias originales, que son lo estimado. Con esto podremos elaborar una escala de error en los distintos tipos de errores posibles (Sec. 4.4). Finalmente, aplicaremos las mismas técnicas de medida usando las secuencias simuladas como estimador, y las originales como valor estimado (Sec. 4.5). Este será el último paso del diagrama de bloque (Fig. 3.1), de donde obtendremos y evaluaremos los resultados.

Error cuadrático medio (*Mean Squared Error*, MSE)

El MSE mide el promedio de los errores al cuadrado, es decir, la diferencia entre el resultado o estimador y lo esperado o estimado. Cuanto más se aproxime el valor MSE a cero más se parecen los valores comparados, es decir que si MSE=0 son idénticos.

[3] *High Frame Rate*

Proporción máxima de señal a ruido (*Peak Signal to Noise Ratio*, PSNR)

El PSNR calcula la relación entre la máxima energía de una señal y el ruido que afecta a su resultado. Cuanto mayor sea el valor PSNR más se pareceran las dos muestras.

Índice de simlitud estructural (*Structural Similarity*, SSIM)

El SSIM sirve para predecir la calidad de imágenes respecto la esperada, es decir, calcular la similitud entre dos imágenes. Los valores serán idénticos si el valor SSIM=1, cuanto más próximo de 1 más parecido siempre en el intervalo $[0, 1]$.

3.3. Implementación

En esta sección describiremos los procesos que usamos para llegar a los resultados esperados, haciendo énfasis en los algoritmos que usamos para cada proceso. Así como en sus parámetros y las conversiones entre formatos que son necesarias para adaptar las entradas de cada proceso.

Los dos procesos que implementamos dentro del marco de trabajo son: la simulación para obtener un flujo de eventos y fotogramas sintéticos y la reconstrucción para obtener una secuencia de vídeo a partir de eventos y fotogramas para poder evaluar los resultados con las técnicas de medida descritas (Sec. 3.2.3)

3.3.1. Simulación

Para realizar la simulación lo primero será escoger el simulador que nos pueda ofrecer un mayor rendimiento de entre los que hemos estudiado en la Sección 2.7. De los simuladores estudiados descartamos Sec. **??**, ya que no implementa la principal característica de los eventos, la asincronía. También descartaremos VSBE (Sec. 2.7.1), ya que no da buenos resultados frente a estímulos rápidos. Los simuladores pyDVS (Sec. 2.7.2) y PIX2NVS (Sec. 2.7.3) aunque tienen un buen rendimiento en la simulación de eventos, no implementan el IMU, elemento importante para muchas aplicaciones de neurorobótica.

Finalmente, de los simuladores descritos en la Sección 2.7, los simuladores 2.7.4, 2.7.6 y ESIM (Sec. 2.7.5) son los que realizan una simulación fidedigna de una cámara de eventos. Usando motores de renderizado para la simulación del movimiento así como una simulación de eventos DVS de calidad. De entre estos tres simuladores, ESIM usa un esquema de muestreo adaptativo [8], que es más eficiente que el muestreo a tasa constante que usan 2.7.4 y 2.7.6. Por lo tanto para el desarrollo de nuestras pruebas usaremos ESIM.

ESIM está desarrollado sobre ROS[4] y dentro de las opciones de renderizado que tiene[5], usaremos la opción de *"Simulating events from a video"*, ya que en este trabajo nos centraremos en la simulación de eventos y fotogramas. Para esto es necesario usar secuencias con una alta tasa de fotogramas por segundo.

[4]ROS: *Robot Operating System* es un sistema operativo que contiene un amplio set de librerías y herramientas de software para el desarrollo de aplicaciones robóticas.

[5]https://github.com/uzh-rpg/rpg_esim/wiki

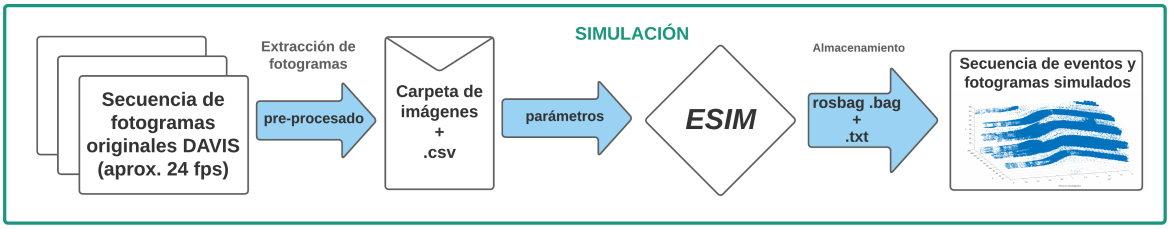


Figura 3.2: Diagrama de bloque de la simulación.

El proceso de simulación está ilustrado en la Figura 3.2. En primer lugar se realiza un pre-procesado de la secuencia de vídeo, obtenida de los fotogramas de la secuencia original DAVIS. El pre-procesado consiste en extraer los fotogramas y crear un archivo *.csv* que los contenga. En segundo lugar se introducen los parámetros de simulación. Algunos como ***data_source***, ***path_to_data_folder*** y ***path_to_output_bag*** hacen referencia a la carpeta de entrada o salida, de otros depende la simulación:

- ***ros_publisher_frame_rate***: fotogramas por segundo del sensor APS simulado.
- ***exposure_time_ms***: tiempo de exposición del sensor APS simulado, en milisegundos.
- ***use_log_image***: '1', para que *ESIM* opere en el dominio de intensidad logarítmica. Cada imagen de entrada (I) será convertida mediante $L = ln(\frac{I}{255+eps})$, donde eps toma el valor del parámetro ***log_eps***, '0.1' por defecto.
- ***contrast_threshold_pos*** y ***contrast_threshold_pos***: Define los valores del umbral positivo y negativo respectivamente, '0.15' por defecto.

Para la simulación con ESIM usaremos los parámetros por defecto. Existe la posibilidad de visualizar los resultados en el renderizador, pero nosotros almacenaremos tanto eventos como fotogramas simulados en un rosbag, *.bag*, y en formato *.txt*. Este es el último paso de la simulación, finalmente tendremos almacenadas en los formatos mencionados las secuencias de eventos y fotogramas sintéticas simuladas a partir de las secuencias originales DAVIS.

3.3.2. Reconstrucción

Para el proceso de reconstrucción de imagen a partir de eventos nos centraremos en los algoritmos basados en síntesis de vídeo, ya que obtienen mejores resultados en la reconstrucción. Y debido a que este tipo de algoritmos usa secuencias de eventos y fotogramas, como nuestras secuencias del *dataset* de prueba. De entre los algoritmos de síntesis de vídeo mencionados en el estado del arte descartamos [54], [57], [60] y [61]; ya que, aunque implementan las funciones básicas de reconstrucción, no obtienen buenos resultados en condiciones adversas, como movimientos bruscos, iluminación insuficiente, etc. [56] obtiene buenos resultados pero es costoso computacionalmente; así como [59] obtiene muy buenos resultados, pero para ello es necesario entrenar su algoritmo de *deep learning*. Por lo que usaremos [55] que obtiene buenos resultados en condiciones adversas gracias al *deblurring* y no tiene tanta complejidad como [57] o [59].

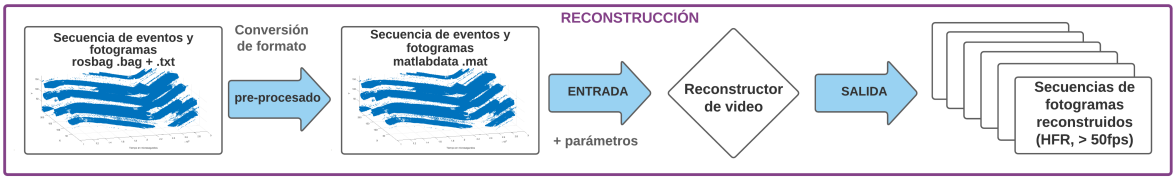


Figura 3.3: Diagrama de bloque de la reconstrucción de vídeo a partir de una secuencia de eventos y fotogramas.

En la Figura 3.3 podemos observar el digrama de bloque detallado del proceso de reconstrucción. Inicialmente tenemos un flujo de eventos y una secuencia de fotogramas contenidos en un archivo *.bag* y la secuencia de eventos en un *.txt*. El reconstructor está implementado sobre Matlab y su entrada debe estar en formato de estructura de Matlab, *.mat*. Por lo tanto es necesario realizar una conversión del formato de las secuencias para adaptarlo a la entrada, que describimos en detalle más adelante. Una vez tenemos las secuencias en una estructura Matlab ya se pueden procesar.

El reconstructor [55] obtiene los eventos filtrando con una ventana en tiempo antes y después del fotograma original. Esta ventana puede desplazarse hacia delante o hacia atrás en el tiempo (con el parámetro *t_ shift*). Se fusionan los eventos formando los fotogramas de pseudo-intensidad y junto con los fotogrmas de intensidad originales obtenemos los fotogramas de intensidad reconstruidos.

El propósito principal del reconstructor [55] es mejorar la calidad en fotogramas con desenfoque de movimiento (*motion blur*), que no es el caso de nuestras secuencias. Por lo que modificaremos los parámetros del reconstructor para nuestras secuencias y haremos una reconstrucción con imágenes de gris medio, los mencionamos más adelante en esta sección y en la Sección 4.3 mostramos las pruebas que hemos realizado para encontrar los valores adecuados para nuestro *dataset* de prueba. Finalmente, se encadenan los fotogramas reconstruidos obteniendo así el vídeo final, en formato *.avi*. En la sección de pre-procesado describiremos el método de reconstrucción con imágenes de gris medio de la cual obtendremos la secuencias de vídeo en imágenes de diferencia, para la futura evaluación de los eventos.

Pre-Procesado

El pre-procesado consiste básicamente en una conversión entre formatos, hemos implementado dos formas de conversión en Matlab, dependiendo de los archivos que usamos:

1. **Rosbag:** Sólo usamos el archivo *.bag* que contiene el flujo de eventos y la secuencia de imágenes por separado. Usamos ROS Toolbox de Matlab para extraer los datos, tomando como tiempo cero la marca de tiempo del primer evento, e introducirlos en la estructura Matlab. Este método resulta muy costoso computacionalmente, sobre todo para la extracción de los eventos, por lo que buscamos un método alternativo.

2. **Rosbag y *.txt*:** La secuencia de imágenes seguimos obteniéndola del *.bag*, mientras que el flujo de eventos lo extraemos del *.txt*. Este método resulta ser más rápido y eficiente que el anterior, por lo que lo usaremos a partir de ahora.

En el caso del método 2 obtenemos los eventos del *.txt*, donde se encuentran la marca de tiempo en segundos ***timeStamp***, la posición $\boldsymbol{x}$ e $\boldsymbol{y}$ y el bit de polaridad ***polarity*** ordenados por columnas. La marca de tiempo la pasaremos a microsegundos y la normalizaremos restándole el

tiempo del primer evento. Al extraer los datos del *.txt* será una simple conversión de una cadena de texto a la estructura de matlab con los tipos *uint32*, *uint16*, *uint16* y *logical* respectivamente, además de contar el número de eventos de la secuencia ***numEvents***.

En el caso de los fotogramas extraemos sus datos del rosbag: marca de tiempo de inicio ***timeStampStart*** dividida en segundos y nanosegundos, que pasaremos a microsegundos, las sumaremos y también normalizaremos con el tiempo del primer evento; la marca de tiempo de fin ***timeStampEnd*** es el tiempo de inicio más el tiempo de exposición; los valores de intensidad de pixel en el rosbag se encuentran en un vector fila, el cual dividiremos en tantos subvectores como filas y tenga el sensor y de longitud el número de columnas x, cada subvector lo introduciremos por filas en un array del tamaño x,y del sensor, formando así los ***samples*** en un *cell* de matlab (en el siguiente párrafo describiremos el pre-procesado para obtener secuencias de vídeo con imágenes en diferencia); y un recuento del número de fotogramas ***numEvents*** El resto de valores son las dimensiones del sensor ***xLength***=240, ***yLength***=180, ***xPosition***=0 e ***yPosition***=0.

En el pre-procesado podemos obtener dos tipos de secuencias a reconstruir: pre-procesado con imagen de intensidad original con los valores de intensidad de pixel del rosbag, descrito en el párrafo anterior y el pre-procesado con imagen de gris medio, consiste en cambiar los valores de ***samples*** por un valor de gris medio (125 en este caso), con el objetivo de obtener secuencias de vídeo con imágenes en diferencia a la salida del reconstructor, en las que observaremos solo los cambios de intensidad de los eventos. Por lo que al final de la reconstrucción tendremos secuencias originales, ruidosas y simuladas reconstruidas mediante imagen original y mediante imagen de gris medio.

Parámetros

- ***option*:** El reconstructor contiene un algoritmo adicional para eliminar el viñeteo, su valor debe ser *2* para activarlo.
- ***dnoise*:** Activa un algoritmo adicional para eliminar ruido, valor a *1*.
- ***timescale*:** Define la escala de tiempo en la que están las marcas de tiempo de los eventos y los fotogramas.
- ***t_ shift*:** Es el desplazamiento o tiempo de cambio que desplaza la ventana de eventos hacia delante o hacia atrás en el tiempo, con el objetivo de obtener el intervalo en el que la imagen sufre el desnfoque de movimiento.
- ***startframe*:** El fotograma de la secuencia de fotogramas con el que queremos empezar a reconstruir.
- ***endframe*:** El último fotograma que usaremos para la reconstrucción.
- ***v_ length*:** Es el número de fotogramas que sintetiza a partir de cada fotograma original.
- ***lambda*:** Un DVS genera un evento cuando detecta que la luminosidad cambia superando el umbral como describimos en la Sección 2.2, y el reconstructor necesita obtener ese valor para la reconstrucción. Para ello esta la ecuación de minimización de energía del reconstructor [55] donde *lambda* es un coeficiente de compensación de la ecuación, para encontrar dicho umbral. El valor de *lambda* influye en la intensidad con la que veremos los objetos tras la reconstrucción.
- ***fps*:** La tasa de fotogramas por segundo del vídeo reconstruido.

4

Pruebas y Resultados

4.1. Introducción

Para realizar pruebas sobre el marco de trabajo descrito en el capítulo anterior (Cap. 3) hemos escogido el simulador ESIM [8] de entre los simuladores disponibles (Sec. 2.7) y el reconstructor [55], basado en *deblurring*, de los reconstructores basados en síntesis de vídeo (Sec. 2.6.8).

Sobre el reconstuctor de vídeo evaluaremos cómo afectan los diferentes tipos de ruidos mencionados en la Sección 3.2.1. También evaluaremos la calidad de simulación de ESIM, mediante la reconstrucción de las secuencias simuladas observaremos el nivel de ruido que presenta el simulador.

4.2. *Dataset* de Prueba

Para realizar estas pruebas escogemos un conjunto de once secuencias obtenidas con una DAVIS 240x180, que provienen del *dataset* [7]. En la Tabla 4.1 podemos ver las secuencias seleccionadas y sus características. Estas secuencias son posteriormente modificadas según describimos en la Sección 3.2. Obteniendo los tres tipos de secuencias: originales, ruidosas y simuladas.

Secuencia	Duración (s)	Nº Eventos	Nº Fotogramas
office_spiral	11.2	6254774	254
office_zigzag	11.9	7735308	248
shapes_6dof	59.7	17962477	1356
shapes_rotation	59.8	23126288	1357
shapes_translation	59.7	17363976	1356
slider_close	6.5	4032668	164
slider_depth	3.4	1078541	87
slider_far	6.4	3442683	137
slider_hdr_close	6.5	3337787	149
slider_hdr_far	6.5	2509582	108
urban	10.7	5359539	278

Tabla 4.1: Secuencias originales escogidas del *dataset* [7].

4.3. Parámetros de Reconstrucción

El reconstructor [55] tiene unos parámetros de ajuste de la reconstrucción, descritos en la Sección 3.3.2. Dado que las secuencias de vídeo ruidosas y simuladasque obtengamos a la salida del reconstructor las compararemos con las secuencias originales, debemos reconstruírlas con los mismos parámetros para que estén en las mismas condiciones. Para ello, mediante las siguientes pruebas sobre los parámetros *t_ shift*, *v_ length* y *lambda*, vamos a buscar los parámetros adecuados para nuestras secuencias. Los resultados obtenidos los evaluaremos visualmente en función de la calidad subjetiva de la reconstrucción.

4.3.1. *t_ shift*

Para nuestras secuencias, ya que no sufren desenfoque de movimiento, buscaremos un valor de *t_ shift* para que la ventana de filtrado de eventos descrita en la Sección 3.3.2 se encuentre centrada en el tiempo con respecto a la imagen original. Para encontrar el valor adecuado de *t_ shift* tomamos distintos valores del parámetro, reconstruimos la secuencia con cada uno de ellos y comparamos visualmente. Hemos tomado los valores en el intervalo [-0.05 , 0.03] a incrementos de 0.005.

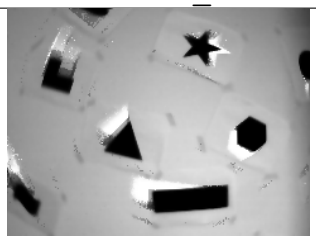
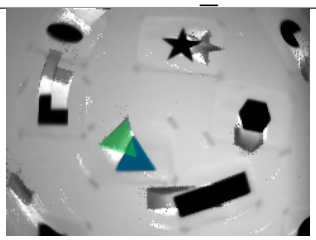
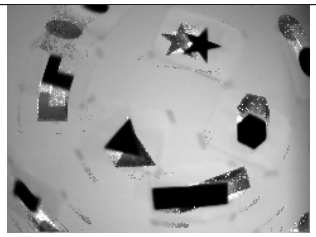
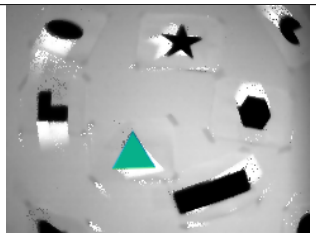
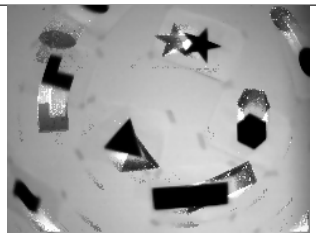
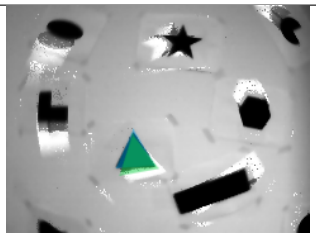
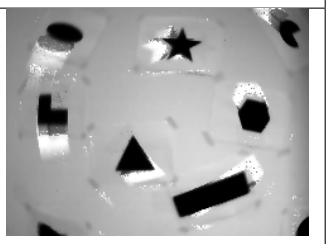
t_ shift	**frame 400_100**	**frame 401_01**	**frame 401_02**
-0.04			
0.015			
0.03			

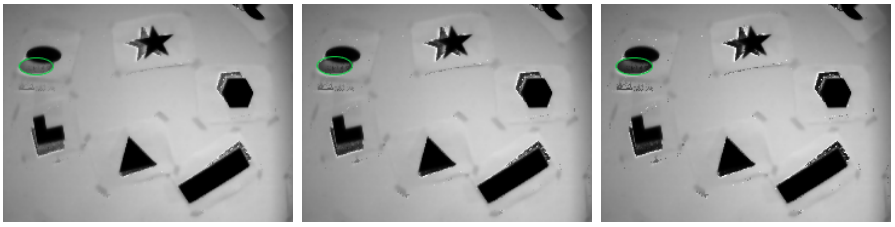
Tabla 4.2: Resultados *t_ shift* para la secuencia shapes_rotation_original. Se observa el cambio de fotograma original de recontrucción (de 400 a 401 en las columnas) para diferentes valores de *t_ shift*. En azul, objetos del fotograma original que se matienen estáticos durante la reconstrucción. En verde, los objetos reconstruidos mediante eventos que deben ser fluidos.

En el caso del desplazamiento su efecto es visible al reconstruir con imágenes originales (Tab. 4.2), donde vemos como se sobrepone los objetos de la imagen original con la reconstruida (por lo que para las siguientes pruebas evaluaremos los resultados con secuencias reconstruidas mediante fotogramas de gris medio, para evitar esta superposición de objetos). El valor de *t_ shift* afecta a la posición donde se encuentran los objetos de los fotogramas reconstruidos (verde) frente a los objetos de los fotogramas originales (azul) durante la reconstrucción, como podemos ver en

la secuencia al cambiar de fotograma de reconstrucción de 400 a 401 en la Tabla 4.2. Para las próximas pruebas utilizaremos $t\_shift = 0{,}015$.

4.3.2. *lambda*

En las pruebas realizadas hemos tomado valores de *lambda* (que por definición es un valor negativo) en el intervalo [-0.5 ,0] con saltos de 0.025. En la Figura 4.1 se puede ver sutilmente en las figuras reconstruidas (borde verde) como para valores de *lambda* pequeños (-0.475, Fig. 4.1c) obtenemos bordes y figuras más definidas, pero también se producen más errores de píxel (como debajo de la figura marcada en verde en la Figura 4.1). Para valores mayores (-0.025, Fig. 4.1a), los contornos se ven más difuminados. Si usamos valores próximos a cero (-0.025) no se produce una buena reconstrucción de la imagen de intensidad, como sucede en la Tabla 4.3 en la fila de arriba para $lambda = -0{,}025$ de la secuencia office_zigzag. Teniendo estos resultados en cuenta, en las próximas pruebas reconstruiremos las secuencias con $lambda = -0{,}2$ (Fig. 4.1b).



(a) -0.025 (b) -0.2 (c) -0.475

Figura 4.1: Resultados *lambda* para el fotograma 402_37 de shapes_rotation_original. Observamos cómo según disminuimos *lambda* la figura reconstruida (marcada en verde) tiene más intensidad

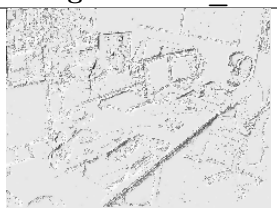
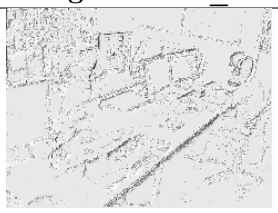
lambda	**fotograma 403_25**	**fotograma 403_26**	**fotograma 403_27**
-0.025			
-0.2			

Tabla 4.3: Resultados *lambda* para office_zigzag_diferencia. Para valores de *lambda* próximos a cero (fila de arriba) se puede producir perdida de intensidad en la reconstrucción.

4.3.3. *v_length*

La cantidad de fotogramas que elijamos reconstruir afectará a la fluidez del vídeo reconstruido (junto con el número de fotogramas por segundo). También afecta notablemente al tiempo de reconstrucción, a mayor número de fotogramas, mayor coste computacional. En las pruebas realizadas sobre distintas secuencias hemos tomado valores de 20 a 200 con saltos de 20.

Usando valores bajos como 20 o 40 observamos como el vídeo no tiene la fluidez deseada (depende también del parámetro *fps* de la reconstrucción, del que dependerá la tasa de fotogramas por segundo del vídeo reconstruido). Los valores altos de 60 para arriba nos proporcionan una fluidez en el vídeo reconstruido, además de permitirnos evaluar las reconstrucción en más puntos (cuantas más imágenes reconstruidas a partir de una original, más evaluaciones entre fotogramas). Pero a partir de 100 no compensa ya que el tiempo de procesamiento crece mucho. Para ello hemos probado calculando las medias de tiempos de reconstrucción de cada fotograma original con v_length=[50,100,150,200] y hemos obtenido medias=[2.02,3.94,6.14,8.377] en segundos, es decir 2 segundos por cada 50 fotogramas aproximadamente, lo que supone un coste computacional muy alto.

Tampoco es recomendable usar valores muy altos, ya que si la ventana de eventos contiene pocos eventos y se intentan reconstruir muchos fotogramas a partir de ellos pueden darse errores de calidad en las imágenes sintetizadas. Por lo tanto nos quedaremos con valores de longitud de reconstrucción (v_length) intermedios de entre 50 y 80, en las próximas pruebas reconstruiremos las secuencias con $v\_length = 50$.

4.4. Influencia del Ruido en la Reconstrucción

Para evaluar cómo afecta el ruido de eventos en la calidad de reconstrucción de imágenes, reconstruiremos las secuencias originales y ruidosas[1] (Sec. 3.2.1) con el reconstructor [55] mediante el proceso descrito en la Sección 3.3.2 y los parámetros escogidos en las pruebas anteriores (Sec. 4.3).

Una vez tenemos todas las secuencias reconstruidas en las mismas condiciones, usaremos las técnicas de medida descritas en la Sección 3.2.3 para evaluar las secuencias ruidosas reconstruidas tomando las secuencias originales reconstruidas como referencia. Estas medidas compararán píxel a píxel entre imágenes de secuencias originales y ruidosas obteniendo un valor de MSE, PSNR y SSIM por cada fotograma de secuencia (Hemos tomado un intervalo de 11 fotogramas originales con v_length=50, son 550 fotogramas reconstruidos). En primer lugar realizamos un análisis de la evolución de la calidad de reconstrucción en el tiempo y posteriormente calculamos la media y la desviación típica de MSE, PSNR y SSIM de cada secuencia para poder comparar entre ellas en función de los porcentajes de ruido.

Las secuencias ruidosas contienen cuatro tipos de ruido introducidos (Sec. 3.2.1), por lo que dividiremos esta sección en cuatro apartados, uno para cada tipo de ruido. Para cada uno de los tipos de ruido tomamos diez porcentajes de ruido (Tab. 4.4), por lo tanto en esta prueba esperamos observar como crece el nivel de error a medida que incrementa el porcentaje de ruido introducido en la secuencia de eventos.

Añadir Eventos	%	50	40	30	25	20	15	10	5	1	0.5
	Coef.	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	0.99	0.995
Eliminar Eventos	%	50	40	30	25	20	15	10	5	1	0.5
	Coef.	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	0.99	0.995
Añadir Ruido en Tiempo	%	5	10	15	20	25	30	40	50	60	70
	Coef.	0.05	0.1	0.15	0.2	0.25	0.3	0.4	0.5	0.6	0.7
Añadir Ruido en Espacio	%	0.25	0.5	0.75	1	1.5	2	2.5	3	4	5
	Coef.	0.0025	0.005	0.0075	0.01	0.015	0.02	0.025	0.03	0.04	0.05

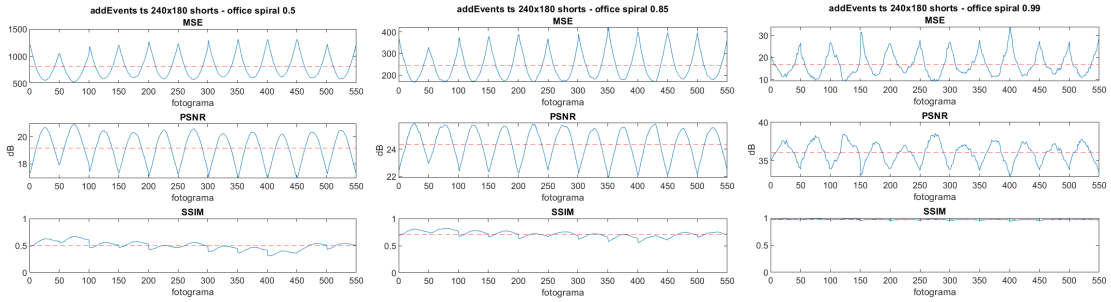
Tabla 4.4: Tipos de ruido, sus porcentajes (%) y sus coeficientes representativos (*Coef.*).

[1]Para esta prueba se han usado las secuencias de vídeo de imágenes de diferencia

4.4.1. Añadir eventos

En la Figura 4.2 observamos la evolución del ruido en tiempo. En la Figura 4.2a se observa cómo con un 50 % de ruido el MSE toma valores alrededor de 900 y SSIM de 0.5, lo que significa un nivel de error alto. En las siguientes gráficas (Fig. 4.2b y Fig. 4.2c) vemos como disminuye el MSE y aumenta SSIM, llegando casi a 1 para con un ruido de un 1 %.

Los valores mínimos de error, que observamos en las gráficas de MSE, PSNR y SSIM, corresponden con la imagen reconstruida a partir de la original. Cuanto más nos alejamos aumenta el error de reconstrucción y diminuye la calidad llegando los picos de máximo error. Los parámetros seleccionados anteriormente (Sec. 4.3) influyen de manera que seleccionando un *t_shift* medio en el tiempo y los valores mínimos se producen en los múltiplos de la mitad de *v_length* (fotograma 25) y los valores máximos en *v_length*, como vemos en las gráficas de la Figura 4.2.



(a) office_spiral_0.5 (b) office_spiral_0.85 (c) office_spiral_0.99

Figura 4.2: Medidas sobre la secuencia *office_spiral* con distintos coeficientes de ruido de adición de eventos.

En la Figura 4.3 podemos ver el resultado de la prueba descrita representado gráficamente, donde observamos como a mayor nivel de ruido (menor coeficiente), mayor error. El MSE medio decrece linealmente al aumentar el coeficiente, mientras que las medias PSNR y SSIM crecen.

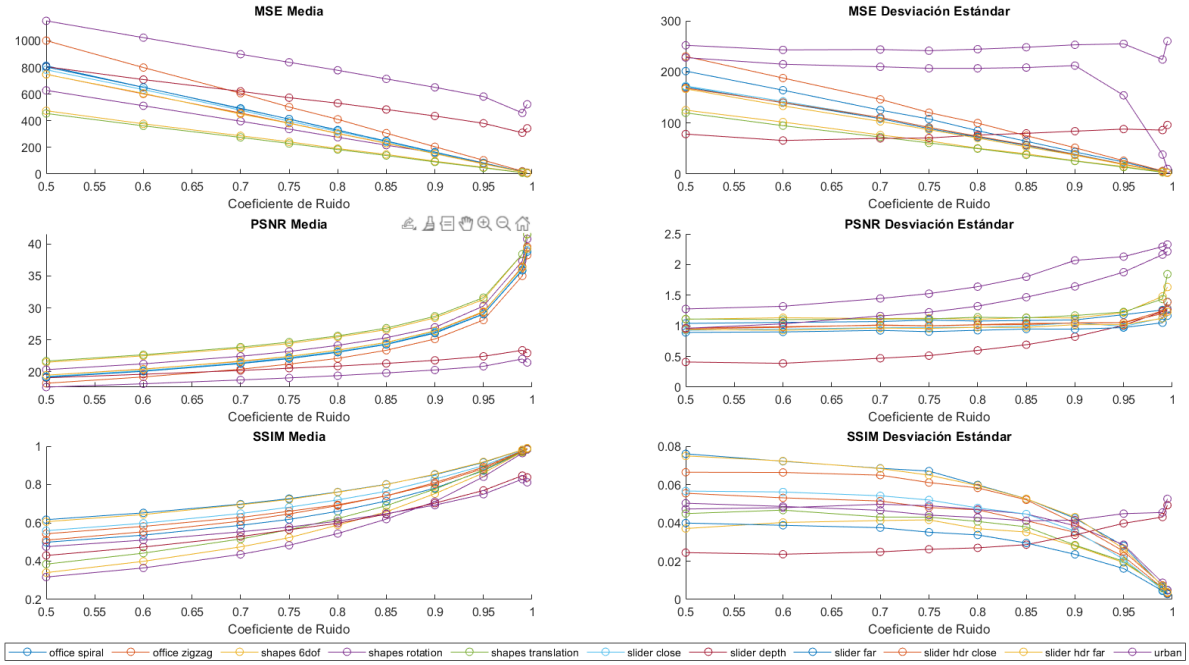
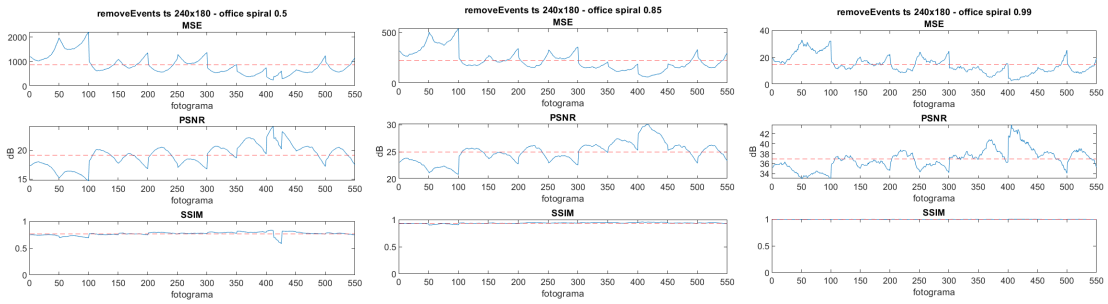


Figura 4.3: Resultados de Ruido de Añadir Eventos.

4.4.2. Eliminar eventos

En la Figura 4.4 vemos la evolución en tiempo del ruido de eliminación de eventos para distintos coeficientes de ruido. Podemos apreciar como eliminando un 50 % de los eventos (Fig. 4.4a) obtenemos MSE alto y SSIM bajo. Mientras que para los coeficientes 0.85 (15 % de eliminación, Fig. 4.4b) MSE decrece y SSIM disminuye y 0.99 (1 % de eliminación, Fig. 4.4c) igual obteniendo menos error aún que con coeficiente 0.85. Comparando estos resultados con los de la Figura 4.2 de ruido de añadir eventos vemos que siguen la misma distribución para MSE y PSNR, mientras que SSIM se ve menos afectado por el ruido de eliminación de eventos.



(a) office_spiral_0.5 (b) office_spiral_0.85 (c) office_spiral_0.99

Figura 4.4: Medidas sobre la secuencia *office_spiral* con distintos coeficientes de ruido de eliminación de eventos.

En la Figura 4.5, comprobamos observamos como el MSE decrece linealmente a medida que disminuye el ruido mientras PSNR y SSIM crecen. Si comparamos estos resultados con los obtenidos del ruido de añadir eventos (Fig. 4.3) observamos que ambos crecen linealmente al aumentar el porcentaje de ruido, y tanto en los porcentajes máximos como mínimos de ruido alcanzan unos valores muy similares. Excepto SSIM, por lo que podemos decir que estructuralmente el reconstructor es más robusto frente al ruido de eliminación de eventos que al de añadir eventos.

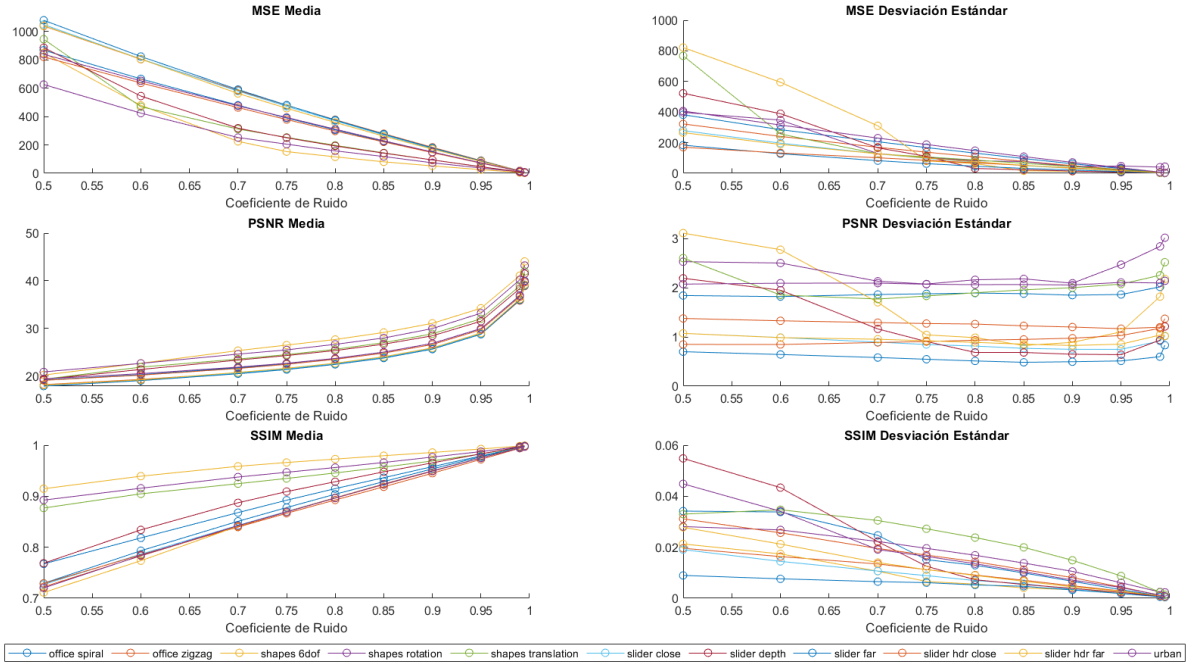
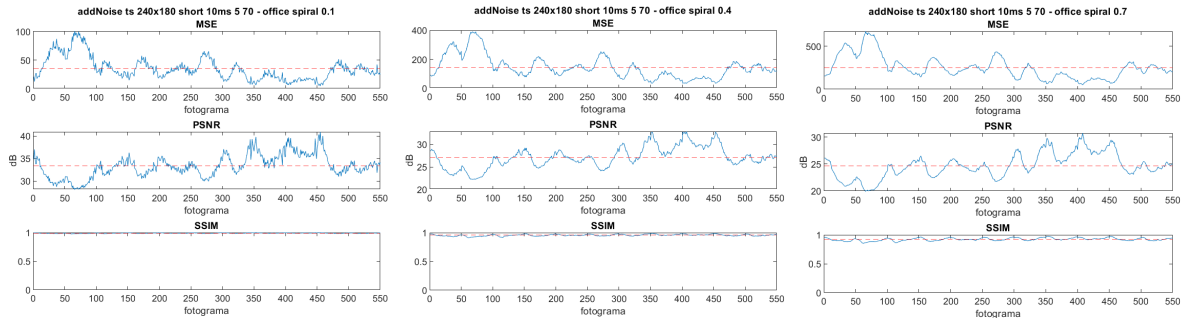


Figura 4.5: Resultados de Ruido de Eliminar Eventos.

4.4.3. Añadir ruido en el tiempo *(ts)*

Las secuencias con ruido en tiempo con distintos porcentajes de desplazamiento de la marca de tiempo, de 5 % a 70 %. En las gráficas de la Figura 4.6 podemos ver como con coeficiente 0.1 (Fig. 4.6a) obtenemos un bajo MSE y PSNR de valor normal, entre 30-40dB. Pero a medida que incrementamos el porcentaje desplazamiento incrementa el error, aumentado el MSE seis veces su valor con coeficiente 0.7 (Fig. 4.6c) y diminuyendo PSNR y SSIM.



(a) office_spiral_0.1 (b) office_spiral_0.4 (c) office_spiral_0.7

Figura 4.6: Medidas sobre la secuencia *office_spiral* con distintos coeficientes de ruido de adición de ruido en tiempo.

En las gráficas de la Figura 4.7 observamos los resultados esperados, dado que al aumentar el coeficiente de ruido vemos como el error incrementa, MSE crece linealmente y decrecen PSNR y SSIM. Comparando estos resultados con los resultados de evolución en tiempo y los resultados globales de los ruidos anteriores (Sec. 4.4.1 y Sec. 4.4.2) observamos que la influencia del ruido temporal es menor que el resto. Aunque tienen una distribución parecida, incrementando linealmente a medida que se incrementa el nivel de ruido.

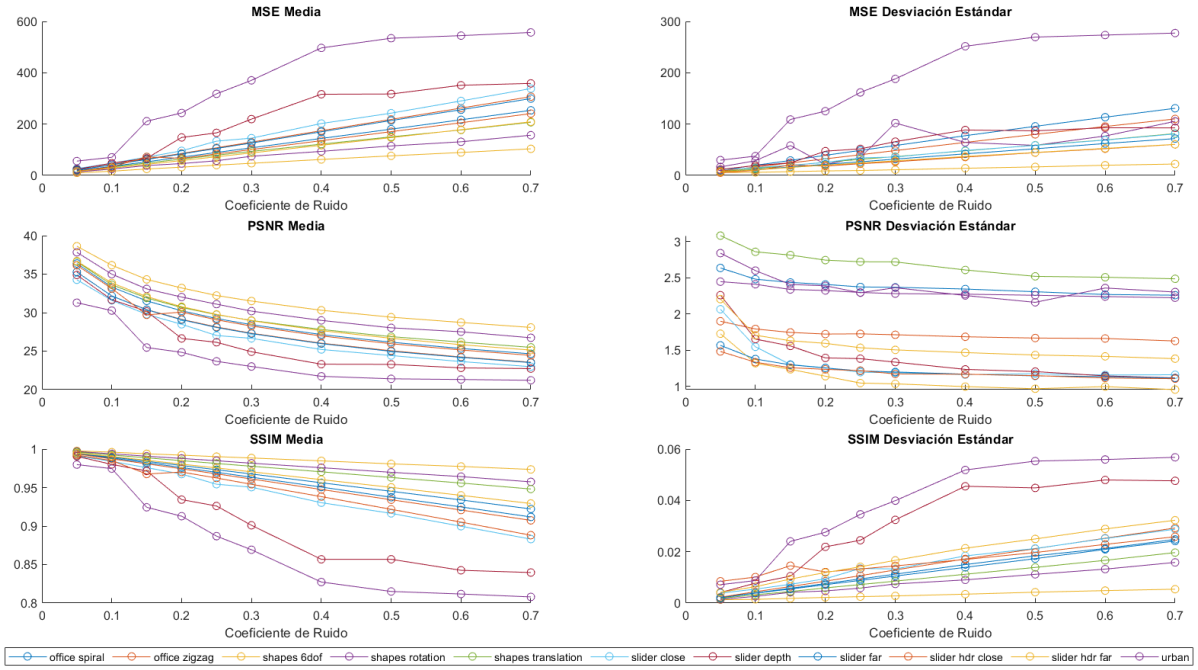
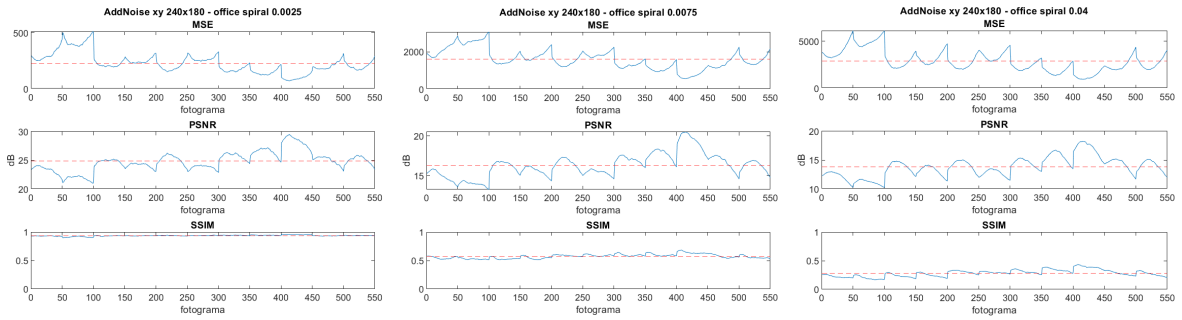


Figura 4.7: Resultados para el Ruido de Añadir Ruido en Tiempo.

4.4.4. Añadir ruido en el espacio *(x,y)*

Al introducir ruido en x,y desplazamos eventos un porcentaje de la diagonal del sensor, de 0.25 % a 5 %. En las gráficas de la Figura 4.8 vemos en detalle la evolución del ruido en tiempo, donde destacamos un gran aumento de error, lo que quiere decir que a mayor ruido introducido la secuencia ruidosa reconstruida se parece menos a la original reconstruida. Comparando este primer análisis con el resto de tipos de ruido observamos que obtiene valores más altos de ruido así como menor simulitud estructural (SSIM).



(a) office_spiral_0.0025 (b) office_spiral_0.0075 (c) office_spiral_0.04

Figura 4.8: Medidas sobre la secuencia *office_spiral* con distintos coeficientes de ruido espacial x,y

En las gráficas de la Figura 4.9 podemos comprobar como a medida que aumentamos el porcentaje de desplazamiento aumenta el error medio por secuencia. Para los porcentajes más bajos el ruido MSE incrementa rápidamente y a partir del 3 % la función sufre una desaceleración. Y como era de esperar PSNR y SSIM decrecen a medida que aumenta el ruido introducido. Con estos resultados y los resultados de los ruidos anteriores podemos concluir que el ruido espacial es el que más afecta a la calidad de la reconstrucción, además de seguir una distribución logarítmica distinta a los otros tipos de ruido que siguen una distribución lineal.

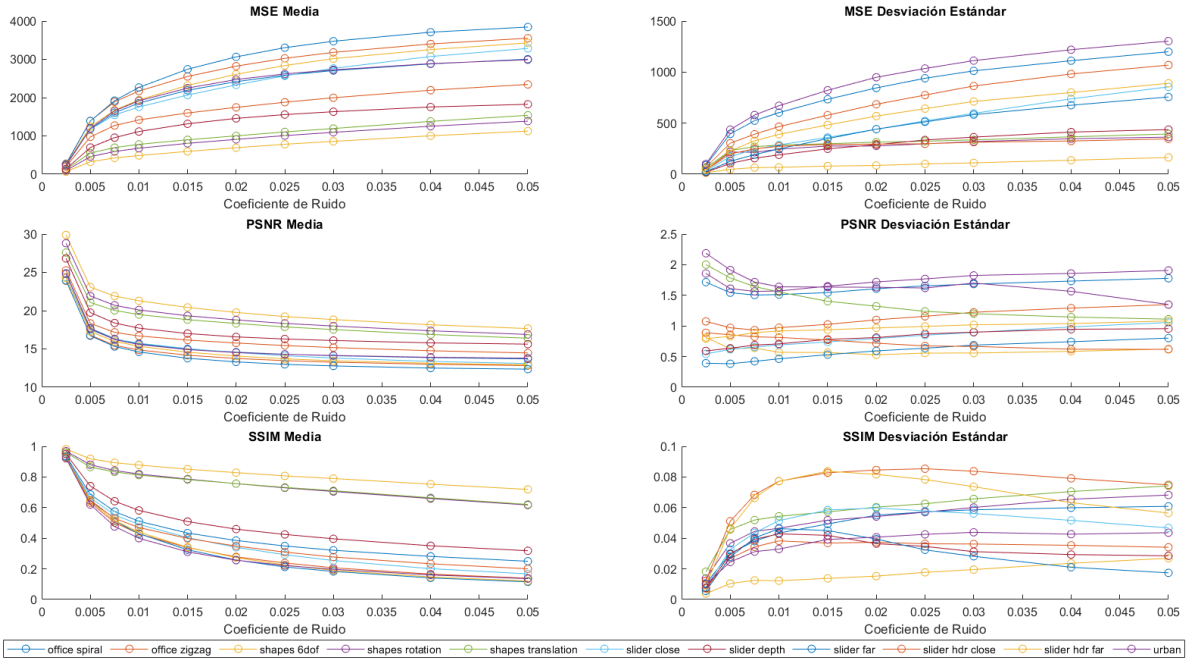
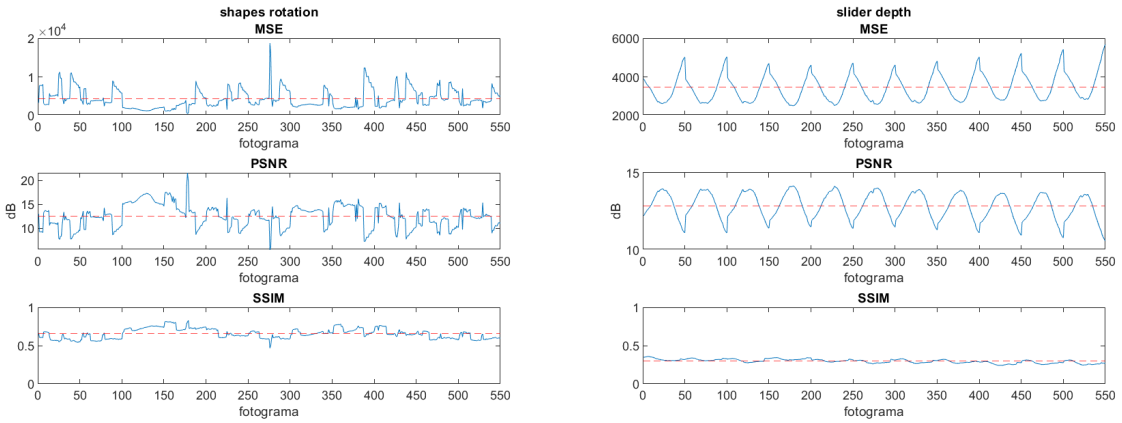


Figura 4.9: Resultados para el Ruido de Añadir Ruido en Espacio.

4.5. Evaluación de la Reconstrucción con Secuencias Simuladas

Para evaluar el proceso de simulación descrito en la Sección 3.3.1, utilizamos las técnicas de medida que mencionamos en la Sección 3.2.3 para realizar una comparación entre las simuladas reconstruidas y las secuencias originales reconstruidas como referencia, con los parámetros definidos en la Sección 4.3. Obtenemos el error medio de las secuencias simuladas comparamos con los niveles de ruido establecidos con las secuencias ruidosas para observar que nivel de distorsión presentan las secuencias simuladas.



(a) shapes_rotation

(b) slider_depth

Figura 4.10: Medidas sobre las secuencias simuladas.

A primera vista podemos ver en las gráficas MSE y PSNR de la Figura 4.10 que el error de las secuencias simuladas es muy alto. También calculamos las medias y desviaciones estándar por cada secuencia y de todas las secuencias juntas en la Tabla 4.5. Comparamos los valores obtenidos en esta tabla con los valores que hemos obtenido para cada uno de los tipos de ruido en las pruebas con las secuencias ruidosas (Sec. 4.4).

Secuencia	MSE		PSNR		SSIM	
	Media	Desv.Est.	Media	Desv.Est.	Media	Desv.Est.
office_spiral	3380.1	1339.7	13.2	1.7	0.371	0.064
office_zigzag	4168.6	1200.6	12.1	1.3	0.235	0.048
shapes_6dof	4428.4	2860.1	12.4	2.5	0.721	0.0418
shapes_rotation	4284.8	2474	12.5	2.4	0.655	0.066
shapes_translation	5908.8	2534	10.8	1.9	0.566	0.109
slider_close	7069.8	1450.5	9.7	0.9	0.137	0.024
slider_depth	3451.8	701.8	12.8	0.9	0.297	0.026
slider_far	7178.9	1335	9.6	0.8	0.098	0.024
slider_hdr_close	5783.5	1186.9	10.6	0.9	0.165	0.076
slider_hdr_far	3552.9	1131.8	12.8	1.3	0.168	0.05
urban	3662.1	1787.8	13	2.2	0.252	0.077
media simulación	4806.3	1636.6	11.8	1.5	0.333	0.055

Tabla 4.5: Medidas sobre las secuencias simuladas. M.=Media, D.E.=Desviación Estándar. Los valores de MSE y PSNR están redondeados al primer decimal y los SSIM al tercero.

En primer lugar comparamos los resultados de las secuencias simuladas (Tabla 4.5) con los resultados del rudio de adición de eventos (Sec. 4.4.1), en concreto con la Figura 4.3 y observamos como el error obtenido en las secuencias simuladas es mucho mayor y sale de escala. Sucede lo mismo cuando comparamos con la Figura 4.7 de las secuencias con ruido de desplazamiento de tiempo (Sec. 4.4.3) y con la Figura 4.5 de las secuencias con eliminación de eventos (Sec. 4.4.2).

Al comparar los resultados obtenidos en la Tabla 4.5 con la Figura 4.9 de ruido de desplazamiento en tiempo (Sec. 4.4.4), observamos que este tipo de ruido genera valores muy altos de error para coeficientes mayores de 0.025. Para verlo con más detalle y compararlo con la media de las secuencias simuladas obtenemos la Figura 4.11 en la cual vemos que las medidas PSNR y SSIM son comparables, pero el MSE de las secuencias simuladas es aún mayor.

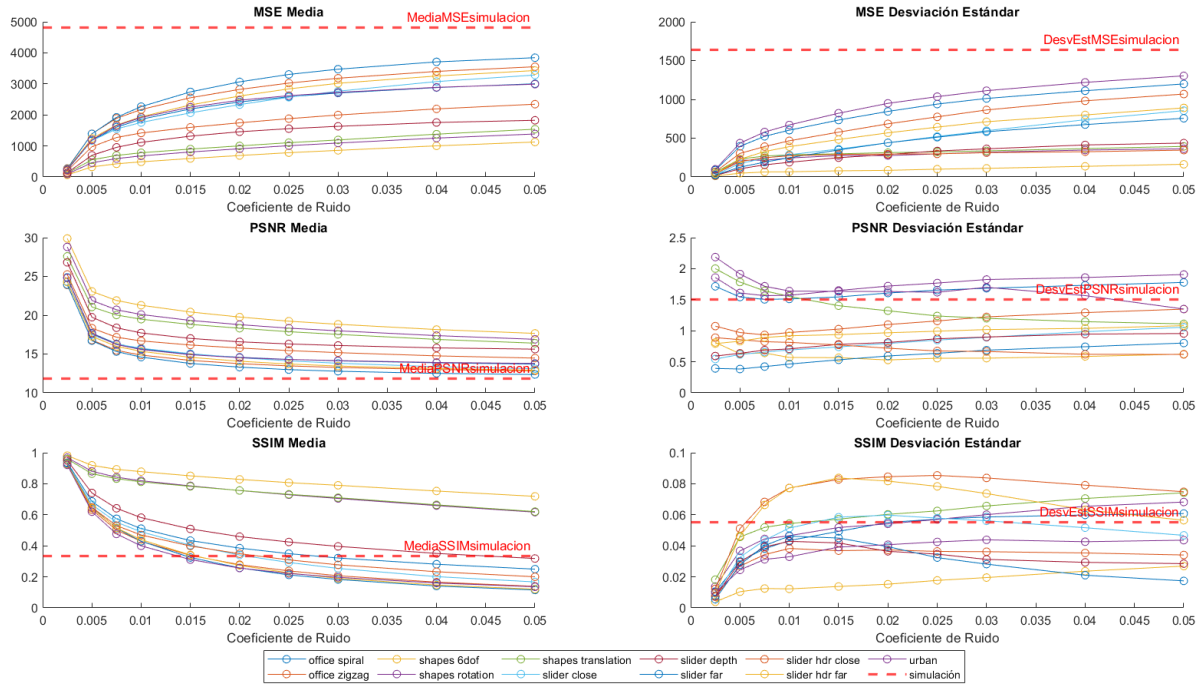
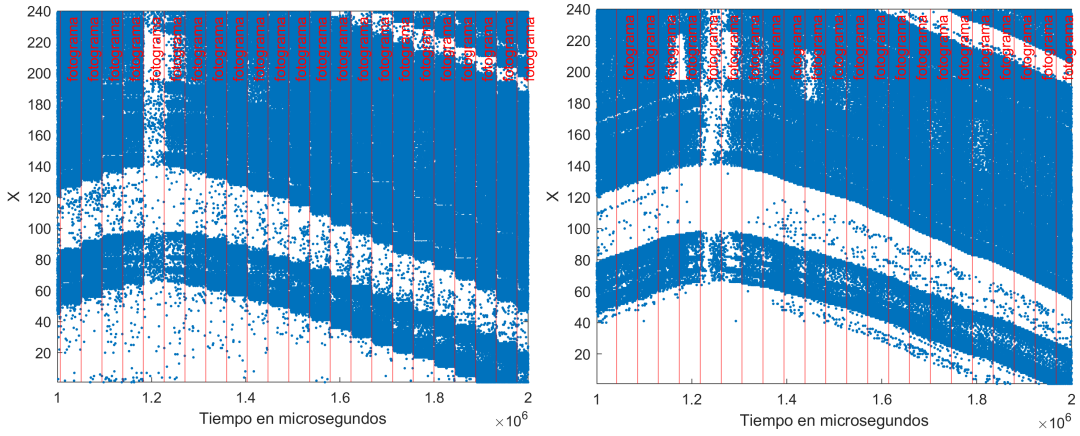


Figura 4.11: Comparación de la simulación con las secuencias con ruido en espacio x,y (Sec. 4.4.4)

Analizando los flujos de eventos sintéticos obtenidos a partir del simulador, hemos observado en la Figura 4.12 el comportamiento de los eventos simulados. Al no hacer una interpolación espacial apropiada entre el fotograma actual y el siguiente, los eventos quedan agrupados en la misma posición hasta que llegue el siguiente fotograma. Por lo que en la Figura 4.12a observamos que falta fluidez en los eventos frente a la original de la Figura 4.12b.

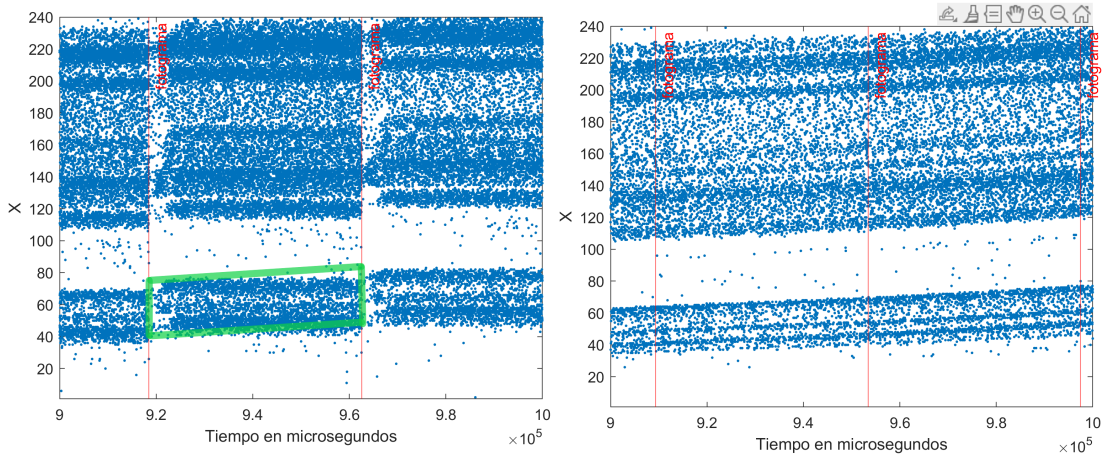


(a) shapes_rotation_simulated 1s

(b) shapes_rotation_original 1s

Figura 4.12: 1 segundo del flujo de eventos de secuencias simuladas (izquierda) y secuencias originales (derecha). La línea vertical roja marca la llegada de un fotograma original a partir del cual se simulan los eventos.

Ampliando la resolución temporal a una décima de segundo y comparando el flujo de eventos simulados en la Figura 4.13a con el flujo de eventos originales 4.13b percibimos más en detalle la falta de fluidez que mencionamos en la figura anterior. Los eventos entre fotogramas quedan agrupados en el mismo pixel hasta la llegada del siguiente fotograma. El recuadro verde en la Figura 4.13a marca donde deberían encontrarse el cojunto de eventos para una buena simulación, más parecida a la secuencia original de la Figura 4.13b.



(a) shapes_rotation_simulated 0.1s.

(b) shapes_rotation_original 0.1s

Figura 4.13: 0.1 segundo del flujo de eventos de secuencias simuladas (izquierda) y secuencias originales (derecha). La línea vertical roja marca la llegada de un fotograma original a partir del cual se simulan los eventos.

Dado que las medidas obtenidas de las secuencias simuladas sobrepasan los niveles de ruido de los cuatro tipos de ruido definidos en la prueba anterior (Sec. 4.4) y visto el comportamiento de los eventos simulados, podemos concluir la prueba diciendo que la simulación descrita en la Sección 3.3.1 no es adecuada para la reconstrucción de vídeo a partir de eventos e imágenes de intensidad tal y como la definimos en la Sección 3.3.2.

5

Conclusiones y Trabajo Futuro

Hemos contribuido desarrollando un marco de trabajo para la reconstrucción de imágenes a partir de cámaras de eventos, por lo que hemos realizado un análisis detallado del estado en el que se encuentra la tecnología de las cámaras de eventos. Posteriormente hemos usado este entorno para realizar las pruebas descritas en el Capítulo 4. Entre ellas un análisis de la influencia del ruido de eventos sobre la reconstrucción de vídeo a partir de síntesis de vídeo, con al que hemos contribuido aportando una escala de niveles de ruido sobre la que evaluar algoritmos de reconstrucción. Y con la evaluación de la reconstrucción con secuencias de eventos simuladas con ESIM 2.7.5, donde hemos aportado un método de evaluación de la calidad de simulación de secuencias para tareas de reconstrucción de imágenes de intensidad.

En cuanto las pruebas realizadas sobre la influencia del ruido en el proceso de reconstrucción de imágenes, hemos probado con cuatro tipos de ruido: añadir eventos, eliminar eventos, ruido temporal y ruido espacial. Hemos observado que los tres primeros afectan parecido a la calidad de la reconstrucción, siguiendo una distribución lineal a medida que aumenta el porcentaje de ruido. Mientras que el ruido espacial, a parte de perjudicar en mayor medida la calidad de la reconstrucción, sigue una distribución logarítmica. También hemos comprobado que el reconstructor es más robusto frente al ruido temporal, aunque estructuralmete también obtiene una calidad aceptable frente a la eliminación de eventos. Mediante estas pruebas hemos elaborado una escala de niveles de ruido sobre la que poder evaluar otros reconstructores, y dado que durante este tiempo han surgido nuevos algoritmos de reconstrucción de imágenes de intensidad basados en síntesis de vídeo [64] [65] [66] que nos abren la puerta a futuros trabajos de reconstrucción. Proponemos el uso del marco de trabajo desarrollado para la evaluación de los nuevos algoritmos de reconstrucción. Además consideramos que el marco de trabajo puede utilizarse para la evaluación de otras tareas de visión artificial de entre las descritas en la Sección 2.6.

Hemos seguido el mismo método de prueba para evaluar el proceso de reconstrucción con secuencias de eventos sintéticos, simuladas con el simulador ESIM. Hemos dado por concluido que el simulador ESIM no es adecuado para la tarea de reconstrucción de vídeo a partir de eventos y fotogramas. Esto se debe a que los eventos simulados se agrupan entre los fotogramas originales sin variar su posición de píxel con el movimiento, perdiendo así la caracterítica fluidez de los eventos originales. Durante la realización de este trabajo han sido publicados nuevos trabajos prometedores sobre la simulación de eventos, tales como: Gehrig et al. [67] y *v2e* [68]. Por tanto proponemos usando el marco de trabajo descrito, realizar una evaluación de estos simuladores y los nuevos que puedan publicar en un futuro. Además de volver a evaluar el simulador ESIM

si se corrigen estos problemas.

Bibliografía

[1] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, A. Taba, B.and Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza. "event-based vision: A survey". 2020.

[2] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "a 128 × 128 120 db 15 μs latency asynchronous temporal contrast vision sensor". *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[3] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. "a 240 × 180 130 db 3 μs latency global shutter spatiotemporal vision sensor". *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[4] Garibaldi Pineda García, Patrick Camilleri, Qian Liu, and Steve Furber. "pydvs: An extensible, real-time dynamic vision sensor emulator using off-the-shelf hardware". pages 1–7, 2016.

[5] M. L. Katz, K. Nikolic, and T. Delbruck. "live demonstration: Behavioural emulation of event-based vision sensors". pages 736–740, 2012.

[6] Yin Bi and Yiannis Andreopoulos. "pix2nvs: Parameterized conversion of pixel-domain video frames to neuromorphic vision streams". pages 1990–1994, 2017.

[7] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. "the event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam". Vol. 36(Issue 2):pages 142–149, Feb. 2017.

[8] H. Rebecq, D. Gehrig, and D. Scaramuzza. "esim: an open event camera simulator". 2018.

[9] Misha Mahowald. "vlsi analogs of neuronal visual processing: A synthesis of form and function". 1992.

[10] K.A. Boahen. "a burst-mode word-serial address-event link-i: transmitter design". *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(7):1269–1280, 2004.

[11] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. "a qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds". *IEEE Journal of Solid-State Circuits*, 46(1):259–275, 2011.

[12] E. Culurciello, R. Etienne-Cummings, and K.A. Boahen. "a biomorphic digital image sensor". *IEEE Journal of Solid-State Circuits*, 38(2):281–294, 2003.

[13] Cho "Dan" and Taejae Lee. "a review of bioinspired vision sensors and their applications". *Sensors and Materials*, 27, 01 2015.

[14] Tobi Delbruck and Carver A. Mead. "time-derivative adaptive silicon photoreceptor array". *Proc. SPIE 1541, Infrared Sensors: Detectors, Electronics, and Signal Processing*, vol. 1541:pp. 92–99., 1991.

[15] Garrick Orchard, Daniel Matolin, Xavier Lagorce, Ryad Benosman, and Christoph Posch. "accelerated frame-free time-encoded multi-step imaging". pages 2644–2647, 2014.

[16] Raphael Berner, Christian Brandli, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. "a 240×180 10mw 12μs latency sparse-output vision sensor for mobile applications". pages C186–C187, 2013.

[17] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. "a qvga 143db dynamic range asynchronous address-event pwm dynamic image sensor with lossless pixel-level video compression". pages 400–401, 2010.

[18] E.R. Fossum. "cmos image sensors: electronic camera-on-a-chip". *IEEE Transactions on Electron Devices*, 44(10):1689–1698, 1997.

[19] Z. Ni, C. Pacoret, R. Benosman, S. Ieng, and S. Régnier. "asynchronous event-based high speed vision for microparticle tracking". page 236–244, 2012.

[20] M. Litzenberger, B. Kohn, A.N. Belbachir, N. Donath, G. Gritsch, H. Garn, C. Posch, and S. Schraml. "estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor". pages 653–658, 2006.

[21] M. Litzenberger, C. Posch, D. Bauer, A.N. Belbachir, P. Schon, B. Kohn, and H. Garn. "embedded vision system for real-time object tracking using an asynchronous transient vision sensor". pages 173–178, 2006.

[22] Zhenjiang Ni, Sio-Hoi Ieng, Christoph Posch, Stéphane Régnier, and Ryad Benosman. "visual tracking using neuromorphic asynchronous event-based cameras". page 925–953, 2015.

[23] David Tedaldi, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza. "feature detection and tracking with the dynamic and active-pixel vision sensor (davis)". pages 1–7, 06 2016.

[24] Daniel Gehrig, Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza. "eklt: Asynchronous photometric feature tracking using events and frames". *International Journal of Computer Vision*, 128:1–18, 03 2020.

[25] Liyuan Pan, Miaomiao Liu, and Richard Hartley. "single image optical flow estimation with an event camera". pages 1669–1678, 06 2020.

[26] Shih-Chii Liu, Bodo Rueckauer, Enea Ceolini, Adrian Huber, and Tobi Delbruck. "event-driven sensing for efficient perception: Vision and audition algorithms". *IEEE Signal Processing Magazine*, 36(6):29–37, 2019.

[27] Federico Paredes-Valles, Kirk Yannick Willehm Scheper, and Guido C. H. E. de Croon. "unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2051–2064, Aug 2020.

[28] Alex Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. "ev-flownet: Self-supervised optical flow estimation for event-based cameras". *Robotics: Science and Systems XIV*, Jun 2018.

[29] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. "unsupervised event-based learning of optical flow, depth, and egomotion". 2018.

[30] Jürgen Kogler, Christoph Sulzbachner, and Wilfried Kubinger. "bio-inspired stereo vision system with silicon retina imagers". *7th ICVS International Conference on Computer Vision Systems*, pages 174–183, 10 2009.

[31] Jürgen Kogler, Christoph Sulzbachner, Martin Humenberger, and Florian Eibensteiner. "address-event based stereo vision with bio-inspired silicon retina imagers". 01 2011.

[32] Stephan Schraml, Ahmed Nabil Belbachir, and Horst Bischof. "an event-driven stereo system for real-time 3-d 360"^{o} panoramic vision". *IEEE Transactions on Industrial Electronics*, 63(1):418–428, 2016.

[33] Stephan Schraml, Ahmed Belbachir, and Horst Bischof. "event-driven stereo matching for real-time 3d panoramic vision". 06 2015.

[34] Yi Zhou, Guillermo Gallego, Henri Rebecq, Laurent Kneip, Hongdong Li, and Davide Scaramuzza. "semi-dense 3d reconstruction with a stereo event camera". *Lecture Notes in Computer Science*, page 242–258, 2018.

[35] Henri Rebecq, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza. "emvs: Event-based multi-view stereo—3d reconstruction with an event camera in real-time". *International Journal of Computer Vision*, 126, 12 2018.

[36] Hanme Kim, Stefan Leutenegger, and Andrew Davison. "real-time 3d reconstruction and 6-dof tracking with an event camera". 9910:349–364, 10 2016.

[37] Christian Brändli, Thomas Mantel, Marco Hutter, Mark Hoepflinger, Raphael Berner, Roland Siegwart, and Tobi Delbruck. "adaptive pulsed laser line extraction for terrain reconstruction using a dynamic vision sensor". *Frontiers in neuroscience*, 7:275, 01 2013.

[38] Elias Mueggler, Basil Huber, and Davide Scaramuzza. "event-based, 6-dof pose tracking for high-speed maneuvers". pages 2761–2768, 2014.

[39] Guillermo Gallego, Jon E.A. Lund, Elias Mueggler, Henri Rebecq, Tobi Delbruck, and Davide Scaramuzza. "event-based, 6-dof camera tracking from photometric depth maps". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(10):2402–2412, 2018.

[40] Samuel Bryner, Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. "event-based, direct camera tracking from a photometric 3d map using nonlinear optimization". pages 325–331, 2019.

[41] Henri Rebecq, Timo Horstschaefer, Guillermo Gallego, and Davide Scaramuzza. "evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time". *IEEE Robotics and Automation Letters*, 2(2):593–600, 2017.

[42] Alex Zihao Zhu, Nikolay Atanasov, and Kostas Daniilidis. "event-based visual inertial odometry". pages 5816–5824, 2017.

[43] Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. "real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization". 09 2017.

[44] Elias Mueggler, Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. "continuous-time visual-inertial odometry for event cameras". *IEEE Transactions on Robotics*, 34(6):1425–1440, Dec 2018.

[45] Xavier Lagorce, Cédric Meyer, Sio-Hoi Ieng, David Filliat, and Ryad Benosman. "asynchronous event-based multikernel algorithm for high-speed visual features tracking". *IEEE Transactions on Neural Networks and Learning Systems*, 26(8):1710–1720, 2015.

[46] Arren Glover and Chiara Bartolozzi. "robust visual tracking with a freely-moving event camera". pages 3769–3776, 09 2017.

[47] Anton Mitrokhin, Cornelia Fermuller, Chethan Parameshwara, and Yiannis Aloimonos. "event-based moving object detection and tracking". *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018.

[48] Timo Stoffregen, Guillermo Gallego, Tom Drummond, Lindsay , and Davide Scaramuzza. "event-based motion segmentation by motion compensation". 2019.

[49] Rafael Serrano-Gotarredona, Matthias Oster, Patrick Lichtsteiner, Alejandro Linares-Barranco, Rafael Paz-Vicente, Francisco Gomez-Rodriguez, Luis Camunas-Mesa, Raphael Berner, Manuel Rivas-Perez, Tobi Delbruck, Shih-Chii Liu, Rodney Douglas, Philipp Hafliger, Gabriel Jimenez-Moreno, Anton Civit Ballcels, Teresa Serrano-Gotarredona, Antonio J. Acosta-Jimenez, and BernabÉ Linares-Barranco. "caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing– learning–actuating system for high-speed visual object recognition and tracking". *IEEE Transactions on Neural Networks*, 20(9):1417–1438, 2009.

[50] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E. Shi, and Ryad B. Benosman. "hots: A hierarchy of event-based time-surfaces for pattern recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1346–1359, 2017.

[51] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. "high speed and high dynamic range video with an event camera". *IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI)*, 2019.

[52] S. Barua, Y. Miyatani, and A. Veeraraghavan. "direct face detection and video reconstruction from event cameras". *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016.

[53] Christian Reinbacher, Gottfried Graber, and Thomas Pock. "real-time intensity-image reconstruction for event cameras using manifold regularisation". 2016.

[54] Christian Brandli, Lorenz Muller, and Tobi Delbruck. "real-time, high-speed video decompression using a frame- and event-based davis sensor". pages 686–689, 2014.

[55] Liyuan Pan, Cedric Scheerlinck, Xin Yu, Richard Hartley, Miaomiao Liu, and Yuchao Dai. "bringing a blurry frame alive at high frame-rate with an event camera". pages 6820–6829, 2019.

[56] Prasan Shedligeri and Kaushik Mitra. "photorealistic image reconstruction from hybrid intensity and event-based sensor". 2019.

[57] Stefano Pini, Guido Borghi, Roberto Vezzani, and Rita Cucchiara. "video synthesis from intensity and event frames". pages 313–323, 2019.

[58] Garrick Orchard, Jie Zhang, Yuanming Suo, Minh Dao, Dzung T. Nguyen, Sang Chin, Christoph Posch, Trac D. Tran, and Ralph Etienne-Cummings. "real time compressive sensing video reconstruction in hardware". *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(3):604–615, 2012.

[59] Zihao W. Wang, Weixin Jiang, Kuan He, Boxin Shi, Aggelos Katsaggelos, and Oliver Cossairt. "event-driven video frame synthesis". 2019.

[60] HC. Liu, FL. Zhang, and D. et al. Marshall. "high-speed video generation with an event camera.". page 749–759, 2017.

[61] Cedric Scheerlinck, Nick Barnes, and Robert Mahony. "continuous-time intensity estimation using event cameras". pages 308–324, 05 2019.

[62] Jacques Kaiser, J. Camilo Vasquez Tieck, Christian Hubschneider, Peter Wolf, Michael Weber, Michael Hoff, Alexander Friedrich, Konrad Wojtasik, Arne Roennau, Ralf Kohlhaas, Rüdiger Dillmann, and J. Marius Zöllner. "towards a framework for end-to-end control of a simulated vehicle with spiking neural networks". pages 127–134, 2016.

[63] Wenbin Li, Sajad Saeedi, John McCormac, Ronald Clark, Dimos Tzoumanikas, Qing Ye, Yuzhong Huang, Rui Tang, and Stefan Leutenegger. "interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset". 2018.

[64] Meng Jiang, Zhou Liu, Bishan Wang, Lei Yu, and Wen Yang. "robust intensity image reconstruction based on event cameras". pages 968–972, 2020.

[65] Limeng Zhang, Hongguang Zhang, Chenyang Zhu, Shasha Guo, Jihua Chen, and Lei Wang. "fine-grained video deblurring with event camera". pages 352–364, 01 2021.

[66] Limeng Zhang, Hongguang Zhang, Jihua Chen, and Lei Wang. "hybrid deblur net: Deep non-uniform deblurring with event camera". *IEEE Access*, 8:148075–148083, 2020.

[67] Daniel Gehrig, Mathias Gehrig, Javier Hidalgo-Carrió, and Davide Scaramuzza. "video to events: Recycling video datasets for event cameras". June 2020.

[68] Tobi Delbruck, Yuhuang Hu, and Zhe He. "V2E: From video frames to realistic DVS event camera streams". *arxiv*, June 2020.